

ProvideX

Version 8.30

Language Reference

Contents	<i>iii</i>
Preface	<i>xiii</i>
Introduction	<i>17</i>
Directives	<i>27</i>
System Functions	<i>389</i>
System Variables	<i>555</i>
Mnemonics	<i>577</i>
System Parameters	<i>653</i>
Control Object Properties	<i>701</i>
Special Files and Devices	<i>737</i>
Special Command Tags	<i>769</i>
Appendix	<i>809</i>
Error Codes and Messages	<i>828</i>
Index	<i>843</i>



ProvideX is a trademark of Sage Software Canada Ltd.

All other products referred to in this document are trademarks or registered trademarks of their respective trademark holders.

©2009 Sage Software Canada Ltd. — Printed in Canada

8920 Woodbine Ave. Suite 400, Markham, Ontario, Canada L3R 9W9

All rights reserved. Reproduction in whole or in part without permission is prohibited.

The capabilities, system requirements and/or compatibility with third-party products described herein are subject to change without notice. Refer to the Sage ProvideX website www.pvx.com for current information.

Publication Release: V8.30

January 20, 2009



Contents

Preface

Using this Documentation	<i>xiii</i>
Conventions	<i>xiv</i>

1. Introduction

About ProvideX	17
Basic Concepts	19

2. Directives

Overview	27
ACCEPT Read Single Keystroke	28
ADD INDEX Add Key to Keyed File	29
ADDR Load & Lock Program in Memory	30
AUTO Automatic Line Generation	31
BEGIN Reset Files and Variables	32
BREAK Immediate Exit of Loop	33
BUTTON Control Button	34
BYE Terminate ProvideX Session	39
CALL Transfer to Subprogram	40
CASE Define Branch Points	42
CHART Control Chart	43
CHECK_BOX Control Check Box	47
CLEAR Reset Variables	54
CLIP_BOARD Use Windows Clipboard	55
CLOSE Close File	56
CONTINUE Initiates Next Iteration of Loop	57
CREATE TABLE Create Keyed File (EFF)	58
CUSTOM_VBX Create/Control VBX	61
CWDIR Change Working Directory	62
DATA Define Data Elements	63
DAY_FORMAT Specify DAY Format	64
DEF CLASS Define Object Class	65

Chapter Outlines	<i>xvi</i>
------------------------	------------

Punctuation/Syntax	25
--------------------------	----

DEF GID/UID Define Group/User ID	67
DEF FN Define Function	68
DEF MSG Define Temporary Message	70
DEF OBJECT Define Object	71
DEF systab= Define System Tables	74
DEF sysvar= Define System Variables	76
DEFAULT Branch If No Matching Case	77
DEFCTL Define/Redefine CTL Values	78
DEFPRN Define as Printer	81
DEFTTY Define Terminal Size	82
DELETE Remove Lines from Program	83
DELETE OBJECT Remove Windows Object	84
DICTIONARY Data Dictionary Access	85
DIM Define Arrays and Strings	86
DIRECT Create File with Keyed Access	89
DIRECTORY Create Subdirectory	91
DISABLE Disable Use of Prefix Table Entry	92
DISABLE CONTROL Disable Control	93
DISABLE EVENT Internal Event Disable	94
DROP Removes Program from Memory	95
DROP_BOX Control Drop Box	96
DROP CLASS Delete Class Definition	102
DROP INDEX Drop Key from Keyed File	103

DROP OBJECT Delete Object	104	LINE_SWITCH Redirect Console Input/Output	175
DROP.ON Drag and Drop	105	LIST List Program Statements.....	176
DUMP Display Variables.....	106	LIST_BOX Control List Box.....	178
EDIT Edit Line in Program	108	LOAD Read Program into Memory	194
ENABLE Re-Enable Use of Prefix Table Entry	110	LOAD CLASS Pre-Load Class Definition.....	195
ENABLE CONTROL Enable Control.....	111	LOAD DATA Load Program Constants.....	196
ENABLE EVENT Internal Event Enable	112	LOCAL Designation of Local Data.....	197
END Halt Program Execution	113	LOCK Reserve File for Exclusive Use.....	200
END DEF End Definition of Multi-line Function ..	114	LONG_FORM Use Long Variable Names.....	201
END SWITCH End Branching of a Program.....	115	MENU_BAR Control Menu Bar.....	202
END WITH End Branching of a Program	116	MERGE Read/Append Lines from File	206
END_IF End IF Directive.....	117	MESSAGE_LIB Establish Message Library.....	208
ENDTRACE End Trace Output	118	MNEMONIC Define File Command Sequence	210
ENTER Specify Arguments.....	119	MSGBOX Display PopUp Message Box.....	212
ERASE Delete File/Directory from System.....	120	MULTI_LINE Control Multi-Line Input	215
ERROR_HANDLER Define Generic Handler	121	MULTI_MEDIA Control Multimedia Interface	223
ESCAPE Interrupt Program Execution	122	NEXT End FOR Loop	225
EXECUTE Execute Basic Instruction	123	NEXT RECORD End SELECT Statement	226
EXIT Terminate Subprogram and Return.....	124	OBTAIN Get Hidden Terminal Input.....	227
EXITTO End Loop, Transfer Control	125	ON EVENT Event Processing	228
EXTRACT Read and Lock Data.....	126	ON..GOSUB Conditional Subroutine Execution ..	230
EXTRACT RECORD Read-Lock Data Record	128	ON..GOTO Conditional Transfer of Control	231
FILE Create New File from File Descriptor.....	130	OPEN Open for Processing	232
FIND Locate and Read Data	131	PASSWORD Apply Password & Encryption.....	239
FIND RECORD Locate & Read Data Record.....	132	PERFORM Call Subprogram, Share Variables.....	243
FLOATING POINT Switch to Scientific Notation ..	133	POP Premature Exit from Stack	245
FOR..NEXT Loop While Incrementing	134	POPUP_MENU Create Popup Menu	246
FUNCTION Declare Object Method	137	PRECISION Change Current Precision.....	248
GET_FILE_BOX Ask for Filename.....	139	PREFIX Set File Search Rules	249
GOSUB Execute Subroutine.....	141	PREINPUT Place Data in Input Queue.....	254
GOTO Transfer within Program	142	PRINT Display Information.....	255
GRID Control Grid	143	PROCESS Call a NOMADS Panel	256
H_SCROLLBAR Control Horizontal Scrollbar	153	PROCESS SERVER Establish Remote Server	258
HIDE Hide Control	156	PROGRAM Create or Assign Program File	259
IF..THEN..ELSE Test Condition	157	PROPERTY Declare Object Properties.....	261
INDEXED Create Indexed File	159	PURGE Clear Data from a File	263
INPUT Get Input from Terminal	160	QUIT Terminate ProvideX Session.....	264
INSERT Insert New Record in File	162	RADIO_BUTTON Control Radio Button	265
INVOKE Execute Operating System Command	163	RANDOMIZE Set Random Key	270
IOLIST Specify Variable List	165	READ Read Data from File	271
KEYED Create Single/Multi-Keyed File	166	READ DATA Read Data from Program.....	273
KEYED LOAD Load and Repair Keyed File	172	READ RECORD Read Record from File	275
LET Assign Value to Variable	173	REDIM Re-Dimension Array	277
LIKE Inherit Properties.....	174	REFILE Clear Data from File	278

RELEASE	Terminate ProvideX Session.....	279
REM	Remark	280
REMOVE	Delete Record from File.....	281
RENAME	Change a File's Name	282
RENAME CLASS	Change Name of Class	283
RENAME CONTROL	Change CTL Values	284
RENAME..INDEX	Rename Keys in Keyed File.....	285
RENUMBER	Change Program Line Numbers	286
REPEAT..UNTIL	Repetitive Execution	287
RESET	Reset Program State	288
RESTORE	Reset Program Data Position	289
RETRY	Re-Execute Failing Instruction.....	290
RETURN	Subroutine/Function Return.....	291
ROUND	Control Rounding.....	293
RUN	Transfer and Execute a Program	294
SAVE	Write Program to File.....	295
SAVE CONTROL	Save Image of Control	296
SAVE DATA	Save Program Constants.....	297
SAVE FILE	Save Bitmap to Disk	298
SELECT..FROM..NEXT RECORD	Query Records	299
SERIAL	Create a Sequential File.....	302
SET_FOCUS	Set Input Focus	304
SET_NBF	Set Number of Keyed I/O Buffers	305
SET_PARAM	Set System Parameters.....	306
SETCTL	GOSUB on CTL Event.....	307
SETDAY	Change Local Date.....	308
SETDEV	Set Device Type Name.....	309
SETDEV IOL=	Alter IOList for Open Channel	310
SETDEV KEY	Alter Keys of Open Channel	311
SETDEV PROGRAM	Set I/O Program	312
SETDEV SEP=	Change File SEP	313
SETDEV TSK()	Add to TSK() List	314
SETDRIVE	Change Default Drive	315
SETERR	Set Error Transfer	316
SETESC	Set Interrupt Processing.....	317
SETFID	Set FID() Definition	320
SETMOUSE	Control/Set Mouse.....	321
SETTIME	Set Local Time	323
SETTRACE	Enable Program Tracing.....	324
SHORT_FORM	Use Short Variable Names	325
SHOW	Show Control	326
SORT	Create File for Sorting.....	327
START	Restart ProvideX	328
STATIC	Add Local Properties at Runtime.....	329
STOP	Halt Program Execution.....	330
SWITCH..CASE	Branch Control	331
SYSTEM_HELP	Invoke Windows Help.....	332
SYSTEM_JRNL	File System Journalization	334
TABLE	Define Translation Table	340
TRANSLATE	Translate Contents of Variable	341
TRISTATE_BOX	Control Tristate Box	344
UNLOCK	Remove Exclusive Use from File.....	349
UNTIL	End REPEAT Loop.....	350
UPDATE	Update Existing Record in File	351
USER_LEX	Define Alternate Keywords	352
VARDROP_BOX	Control Variable Drop Box	354
VARLIST_BOX	Control Variable List Box	360
V_SCROLLBAR	Control Vertical Scrollbar	365
VIA	Assign Variable Indirectly.....	368
VIDEO_PALETTE	Control Video Colours.....	370
WAIT	Temporarily Halt Execution.....	372
WAIT FOR EVENT	Wait for Event	373
WEND	End WHILE Loop	374
WHILE..WEND	Repeat Statements	375
WINPRT_SETUP	Windows Printer Setup	376
WITH..END WITH	Object Reference Construct	382
WRITE	Add/Update Data in File	383
WRITE RECORD	Write Record	386

3. System Functions

Overview	389
@()	Location Function	390
@X() / @Y()	Convert X/Y Coordinates	391
ABS()	Absolute Value	392
ACS()	Return Arc-Cosine	393
AND()	Logical AND	394
ARG()	Command-Line Argument	395
ASC()	Get Internal Character Value.....	396
ASN()	Returns Arc-Sine Function	397
ATH()	Convert Hex	398
ATN()	Return Arc-Tangent	399
BIN()	Binary String from Numeric Value.....	400
BSZ()	Bank Memory Size	401
CHG()	Notify if Variable Has Changed	402

CHR()	ASCII Character of Value	403	KEY()	Return Key of Next Record	470
CMP()	Compress Data	404	KGNG()	Generate Record Key	471
COS()	Return Cosine	406	LCS()	Return Lowercase String	472
CPL()	Compile String	407	LEN()	Return String Length	473
CRC()	Cyclic-Redundancy-Check	408	LNO()	Return Line Number	474
CSE()	Case Compare	409	LOG()	Return Base 10 Logarithm	475
CTL()	Return CTL Definition	410	LRC()	Longitudinal-Redundancy Check	476
CVS()	Convert String	412	LST()	Return List Form of Statement	477
DEC()	Get Binary of String	414	MAX()	Return Maximum Value	478
DIM()	Generate String/Get Array Size	415	MEM()	Return Memory Value	479
DIR()	Get Current Directory	417	MID()	Return Substring	480
DLL()	Call Windows DLL	418	MIN()	Return Minimum Value	481
DSK()	Get Current Disk Drive	421	MNM()	Return Mnemonic Value	482
DTE()	Convert Date	422	MOD()	Return Modulus	483
ENV()	Get Environment Values	424	MSG()	Return Message Text	484
EPT()	Return Exponent Value	426	MSK()	Scan String for Mask	486
ERR()	Test Error Value	427	MXC()/MXL()	Return Maximum Column/Line	488
EVN()	Evaluate Numeric Expression	429	NEW()	Create New Object	489
EVS()	Evaluate String Expression	430	NOT()	Invert String Bits/Logical Condition	490
EXP()	Raise to Base Ten	431	NUL()	Return Test for Null	491
FFN()	Find File Number	432	NUM()	Convert String to Value	492
FIB()	Return File Information Block	434	OBJ()	Return Object Information	493
FID()	Return File Information Descriptor	438	OPT()	Return File Open Options	495
FIN()	Return File Information	441	PAD()	Pad/Truncate String	496
FPT()	Return Fractional Part	445	PACK()	Pack Numeric Data	498
GAP()	Return Odd Parity String	446	PFX()	Return Prefix Value	499
GBL()	Reference Global String Variable	447	PGM()	Return Program Line	500
GEP()	Return Even Parity String	449	POS()	Scan String	502
HSA()	Highest Sector Available	450	PRC()	Round Number to Precision	503
HSH()	Generate Modified Value	451	PRM()	Return Parameter Value	504
HTA()	Get Hex Value of String	454	PTH()	Return Pathname	506
HWN()	Highest Unused Window Number	455	PUB()	List Public Programs	507
I3E()	Convert to/from IEEE Format	456	RCD()	Return Next Record	508
IND()	Return Next Record Index	457	RDX()	Convert ASCII to Radix-40	509
INT()	Return Integer Portion	458	REC()	Expand IOList Specification	510
IOL()	Get IOList Specification	459	REF()	Control Reference Count	512
IOR()	Logical OR	460	RND()	Return Random Number	513
JST()	Justify String	461	RNO()	Return Next Record Number	514
JUL()	Return Julian Date	463	SEP()	Return Field Separator	515
KEC()	Return Key of Current Record	465	SGN()	Return Sign of Value	516
KEF()	Return First Key of File	466	SIN()	Sine Function	517
KEL()	Return Last Key of File	467	SQR()	Square Root	518
KEN()	Return Key After Next	468	SRT()	Sort String	519
KEP()	Return Prior Record's Key	469	SSZ()	Return Sector Size	521

STK() Program Call Stack 522
STP() Strip Leading/Trailing Characters..... 523
STR() Convert Numeric to String 525
SUB() Substitute Text 527
SWP() Swap Data 528
SYS() Invoke Operating System Command 529
TAN() Return Tangent 531
TBL() Convert String Via Table..... 532
TCB() Return Task Information 534
TMR() Timer..... 541
TRX() Convert Radix-40 to ASCII..... 542

TSK() Returns Entry from Task List..... 543
TXH() Text Height 544
TXW() Text Width 545
UCS() Return Upper Case String..... 546
UCP() UnCompress Data 547
UPK() Unpack Numeric Data 548
VIN()/VIS() Obtain Value of Variable..... 549
XEQ() In-line Subprogram Execute 551
XFA() Extended Field Attributes..... 552
XOR() Logical Exclusive OR 554

4. System Variables

Overview 555
BKG Background Process Status 556
CHN Channels Open 556
CTL Control Signal Code..... 557
DAY Return Current System Date 557
DLM Return System Directory Delimiter 558
DSZ Data Space Size Available to User 559
EOM End of Message Character String 559
ERR Last System-Detected Error Value 560
ERS Line Number of Last Error 560
ESC ASCII ESCape Character 561
GFN Highest Available Global Channel 561
GID Operating System Process Identifier..... 562
HFN Highest Available Local Channel 562
HLP Last Specified HLP= Value..... 563
HWD Starting/Home Directory..... 563
LFA Last File Number Accessed..... 563
LFO Last File Number Opened..... 564
LIP Input Location: Column, Line 564
LPG Lead Program Name 564
LWD Current Working Directory 565
MSE Mouse State 565

MSL Length of String Matching Last MSK 567
NAR Number of Arguments, Start ProvideX 567
NID Network or Network Node ID..... 567
PFX Current Prefix Setting..... 568
PGN Current Program Pathname 568
PRC Precision Currently In Effect 569
PRM ProvideX Parameter Settings 570
PSZ Current Program Size..... 570
QUO ASCII Quote Character 571
RET Operating System's Last Error Code..... 571
RND Random Number Generator..... 571
SEP ProvideX Field Delimiter..... 572
SID System Identification Code..... 572
SSN System Software Identifier 572
SYS Operating System Identification 573
TIM Time in Hours Past Midnight 573
TME Time in Hours Past Midnight 574
TMS Seconds Expired in Current Minute 574
TSM Error Status of Current Program 575
UID Current UserID..... 575
UNT Lowest Available Local Channel 576
WHO Current UserID 576

5. Mnemonics

Overview 577
Dynamic Information in Mnemonics 580
Mnemonic Categories 581
List of Mnemonics 585
'@@' Define Cursor Position Sequence 585
'+&' & '-\$' For Internal Use Only 585

'2D' Use 2D Controls 585
'3D' Use 3D Controls 586
'4D' Use 4D Controls 586
'AB' Abort (For Windows Spooler)..... 586
'ARC' Define/Draw Arc 586
'AT' Character Attribute Output Sequence..... 587

'+B' & '-B' Output Buffering On/Off	587	'+E' & '-E' Multi-line Enter as Tab	603
'Bn' Background Colour	588	'EB' End Blinking Mode (DOS)	603
'BACKGR' or 'BK' Next Colour Is Background	588	'EE' End Echo Mode	603
'BB' Begin Blinking	588	'EF' End Expanded Print	603
'BE' Begin Echoing	589	'EG' End Generating Error #29	603
'BEEP' Simple Sound Effect	589	'EI' End Input Transparency	604
'BG' Begin Generating Error #29	589	'EJ' End Box Joining	604
'BI' Begin Input Transparency	590	'EL' Start Edit Key Load	604
'BJ' Join Box Intersections	590	'EL' End VFU Load	604
'BK' Next Colour Is Background	590	'EM' End Output Markup Mode	605
'BLACK' & '_BLACK' Colour Text	591	'EO' End Output Transparency	605
'BLUE' & '_BLUE' Colour Text	591	'EP' Start Expanded Print	605
'BM' Begin Output of Markup Files	591	'ER' End Reverse Video	605
'BO' Begin Output Transparency	591	'ES' Send Escape	606
'BOX' Define / Draw a Box	592	'ET' End Type Ahead	606
'BR' Begin Reverse Video	592	'EU' End Underscoring	606
'BS' Cursor Back One Space	592	'EW' End WrapAround	606
'BT' Begin Type-Ahead Mode	593	'+F' & '-F' Signal Change of Focus On/Off	606
'BU' Begin Underscoring	593	'Fn' Foreground Colour	607
'BW' Begin WrapAround	593	'FF' Form Feed	607
'BX' Define / Draw a Box	593	'FILL' Define Fill Style	607
'*C' Automatic Output on CLOSE	594	'FL' Start Function Key Load	608
'Cn' Control Cursor Display Mode	594	'FONT' Define / List Fonts	609
'CAPTION' Replace Caption for Window	594	'FRAME' Define / Draw a Frame	610
'CE' Clear from Cursor to End of Screen	595	'GD' Define Graphics Character Set	611
'CF' Clear Foreground Mode	595	'GE' End Graphics Data	611
'CH' Position Cursor at Home	595	'GF' Default Font for Window Objects	612
'CI' Clear Input Type-Ahead Buffer	595	'GOTO' or 'WG' Make Window Current	612
'CIRCLE' Define / Draw a Circle	596	'GREEN' & '_GREEN' Colour Text	612
'CL' Clear from Cursor to End of Line	596	'GS' Start Graphics Data Transmission	613
'COLOUR' & '_COLOUR' User-Defined Colours	596	'*H' Control Screen Colours	613
'CP' Condense Print for Screen	597	'HIDE' Control Window Display	614
'CPI' Logical Characters per Inch	597	'*I' Input Conversion Table	614
'CR' Carriage Return	598	'+I' & '-I' Implied Decimals On/Off	614
'CS' Clear Screen	598	'IC' Insert a Space at Cursor	614
'CURSOR' Control Cursor, Mouse Pointer	598	'IMAGE' Define a Graphics Group	615
'CYAN' & '_CYAN' Colour Text	599	'JC' Justify Centre	616
'+D' & '-D' Obsolete	599	'JD' Justify Decimal-Aligned	616
'DC' Delete Character at Cursor	599	'JL' Left-Justify Text	616
'DEFAULT' or 'DF' Define Default	600	'JN' Right-Justify for Numeric	616
'DIALOGUE' Define/Draw Dialogue Region	600	'JR' Right-Justify Text	616
'DN' Move Cursor Down a Line	602	'JS' Left-Justify String	616
'DO' Delete Objects in Scroll Region	602	'L6' Set to 6 LPI	617
'DROP' or 'WD' Drop Identified Window	602	'L8' Set to 8 LPI	617

'LC' Mixed-Case User Input	617	'Sn' Slew to Channel	637
'LD' Delete Current Line	617	'SB' Set Mode to Background	638
'LF' Line Feed (Advance Line)	617	'SCROLL' Define/Control Scroll Region	638
'LI' Insert Line	617	'SE' & 'SD' Scroll Enable/Disable	638
'LINE' Define / Draw a Line	618	'SF' Set Mode to Foreground	639
'LM' Landscape Mode	618	'SHOW' / 'HIDE' Control Window Display	639
'LPI' Logical Lines / Inch	618	'SIZE' Control Visual Size of Window	639
'LT' Move Left One Column	619	'SL' Start VFU Load	640
'MAGENTA' & '_MAGENTA' Colour Text	619	'SN' Native Screen Mode	640
'MAXSIZE' & 'MINSIZE' Window Resize Limit	619	'SP' Standard Print	640
'ME' Begin Edit Mode	620	'SR' Scroll Reset	641
'MESSAGE' Define Message Bar Text	620	'SWAP' or 'WS' Swap Windows on Stack	641
'MINSIZE' Window Resize User Limit	621	'SX' Set Extended Screen Mode	641
'MN' End Edit Mode	622	'+T' & '-T' Text Display On/Off	641
'MODE' Set Attributes and Colour	622	'TEXT' Draw Text	642
'MOVE' or 'WM' Relocate Current Window	623	'TEXTWDW' Create Text Window	643
'MP' Print Mode (Parallel)	623	'TR' Terminal Read from Start	643
'MS' Print Mode (Serial)	623	'TW' Transmit Windows as String	644
'+N' & '-N' Control Drop/List Box Write Error	623	'+U' & '-U' Screen Refresh On/Off	644
'NI' Next Input Numeric	624	'UC' Convert Input to Upper Case	644
'*O' Output Conversion Table	624	'UP' Move Up One Line	644
'OPTION' On-The-Fly Setting	624	'+V' & '-V' Control Row Highlighting	645
'OFFSET' Offset for *WINPRT*	629	'VT' Slew to S6, Vertical Tab	645
'+P' & '-P' Define Mouse Movement	630	'!W' For Internal Use Only	645
'PE' Auxiliary Port Off	630	'+W' & '-W' Windows-Style Windows	645
'PEN' Define Pen Style	630	'WA' Define / Draw Window	646
'PICTURE' Define / Draw Picture	631	'WC' Save/Copy Current Window	646
'PIE' Define / Draw Pie Slice	632	'WD' Drop Identified Window	646
'PM' Portrait Mode	633	'WG' Make Window Current	646
'POLYGON' Define/Draw a Polygon	633	'WHITE' & '_WHITE' Color Text	646
'POP' or 'WR' Restore Previous Window	633	'WINDOW' or 'WA' Define / Draw Window	647
'PS' Auxiliary Port On	634	'WM' Relocate Current Window	648
'PUSH' or 'WC' Save/Copy Current Window	634	'WP' Wide Printer (DOS)	648
'*R' OS Command String	634	'WR' Remove Current Window	648
'RB' Ring Bell	634	'WRAP' WrapAround On/Off	648
'RC' Return Cursor Address	635	'WS' Swap Windows On Stack	648
'RECTANGLE' Draw a Rectangle	635	'WX' Windows Definition Sequence	649
'RED' & '_RED' Colour Text	635	'*X' Program to Call on CLOSE	649
'RL' Return Line Contents	636	'+X' & '-X' Windows 'X' Close Button	650
'RM' Reset to Default Mode	636	'XP' Line Mode (DOS)	650
'RP' Terminal Read to End	636	'YELLOW' & '_YELLOW' Colour Text	650
'RS' Restore Screen	636	'+Z' & '-Z' Text Mode Like Windows	650
'RT' Move Right One Column	637	'ZX' Return Attributes as per BBx	651
'+S' & '-S' Substitute Solid Lines On/Off	637		

6. System Parameters

Overview	653	'FE' Obsolete	665
List of System Parameters	655	'FF'= File Format	666
'1U' Force Dedicated User Slot	655	'FI' Ignore Format Mask Error	666
'3D' 3D in Windows	655	'FL' Filename in Lower Case	666
'AD' Auto-DIM Array	655	'FN' Filename As-Is: No Case Conversion	667
'AH' Alternative 'WINDOW'/'BOX' Heading	656	'FO'= Format Overflow Character	667
'AI'= Automatic Line-Number Increment	656	'FP' Floating Point	667
'AP' Auto-Enable PDF Output	656	'FS'= Default Field Separator	667
'AW' Alternate WINPRT_SETUP	656	'FT' Trapping the F10 Key	668
'BO' Base Zero for Level / Window	656	'FU' Filename in Upper Case	668
'BF'= Common File Buffers	657	'FX' Force EXTRACT	668
'BL'= Break Lines in Listings	657	'HC' Obsolete	668
'BT' Binary Test: 1st Read	657	'HP' LibHaru *PDF*	669
'BX' BBx Emulation	658	'IO' Ignore Null Substring (No Error 47)	669
'BY'= Base year	658	'I2' Ignore Max. Record Count (No Error 2)	669
'CD' Check Current Directory	658	'IC' Ignore Case	669
'CE' Obsolete	658	'IM' Insert Mode for Input	669
'CF' Bypass Console Flush	658	'IR' Insert Mode Reset (Decimal Point)	670
'CH'= Hover Colour	659	'IS'= CTL for Input Ending on SIZ=	670
'CI' Cache IOList	659	'IW' Terminate Invoke Wait	670
'CO'= Mouse Over Colour	659	'IZ'= Ignore Max. Memory Setting	670
'CS' Coloured Syntax	660	'JC' Obsolete	670
'CT'= Character Time-out	660	'KF'= Keyed File Format	671
'CU'= Currency Symbol	660	'KR' Keyed File I/O Emulates BBx	671
'DO' Divide by Zero	660	'LB'= Colour for Line # in Break Points	671
'DB'= Dynamic File Buffers	660	'LC' List Variables in Lower Case	671
'DC' Destructive Cursor	661	'LD' List Directives in Lower Case	672
'DD' Convert Directory Delimiter	661	'LE' SAVE / LIST Indent Statements	672
'DF'= Enforced Delay Time after 'FF'	661	'LF' Long Form Variables	672
'DL'= Enforced Delay Time after 'LF'	662	'LM' List, Show Matched Strings	672
'DP'= Decimal Point Symbol	662	'LP' Obsolete	672
'DT'= Device Time-out	662	'LS'= Colour for Line with Syntax Error	673
'DW'= Delay Time after 'WI'	663	'LU' Lock Unnecessary: Serial Files	673
'EG' End Generation of Error #29	663	'LW' For Internal Use Only	673
'EL'= Encryption Level	663	'LZ' Suppress Leading Zeros	673
'EO' Embedded 'EO' Mnemonics	664	'MB'= MegaBytes: File Segment Size	674
'ES' Display OS Errors in Command Mode	664	'MC' Maintain Case	674
'EX' Apply Execute at Level 0	664	'MF'= Multi-Line Size Factor	675
'F' Suppress Commas on Numeric Overflow	665	'MP' Returns Positive Modulus Value	675
'F4' Return CTL=4 for Exit	665	'MS'= Memory for Program Swap	675
'FB'= Dedicated File Buffers	665	'MX' User-Defined Message Box	675
'FC' Force File Commit	665	'NE' Subprogram Error Report	676

'NI' Ignore Blanks in Numeric Fields	676	'SL'= Save Command Lines	687
'NK' Null Key Stripping	676	'SP' Set Printer Default	687
'NL' Suppress LET Directive in Listings	676	'SR' Small Reads	687
'NN' No Line Numbers as References	676	'SS' Check Structure on Save	688
'NR' No Intermediate Rounding on Division	676	'SV'= Generate for Older Version	688
'NS' No Swapping	677	'SW'= Scroll Wheel	688
'NX' Obsolete	677	'SZ'= Maximum Memory Size for Session	688
'OC' Commit Prior to OPEN Directive	677	'TA'= Turbo Mode Acknowledgement	689
'OF'= Maximum Size Before Output Flush	677	'TB' Toolbar Size	689
'OL'= Maximum Buffers for OPEN LOAD	677	'TC'= Tip Colour	689
'OM' Old Style Mask	678	'TH'= Thousands Separator	690
'OP' Return Original Program Name	678	'TL' LIKE Emulates Thoroughbred	690
'OR' Full OS Path for Rename	678	'TN' Strip Trailing Nulls	690
'OW'= Owner Application Code	678	'TT'= Timed Trace	691
'PC'= Program Load Caching	678	'TU' Thin-Client Turbo Mode	691
'PD'= Default Precision for Current Session	679	'TX' Default String-Template Field Separator	691
'PE' Password Error Control	679	'UL' Un-Numbered Line Assignment	692
'PF' EMS Page Frame	679	'UM' Upper Memory Blocks	692
'PL'= Program Libraries	679	'VC' VT100 Cursor Mode Line Wrap	692
'PO' Path Original	680	'VM' Direct Memory Addressing	692
'PP' Prompt for Password	680	'VP'= Variable Pitch	692
'PQ' Password Queue	680	'VR'= Verify Read	693
'PS'= Maximum Program Size (KB)	681	'VW'= Verify Write	693
'PT' Obsolete	681	'WB' WindX BREAK Recognition	693
'PU' Upper-Case Prefix	681	'WD'= Defer File Writes	694
'PW'= Password Character for Multi-Line	681	'WF' Force Windows Screen Update	694
'PZ' Suppress Program Size Warning	681	'WH'= Delay Retry: Locking File Headers	694
'Q_'= Lowest Task Priority	682	'WI'= Windows Instruction Count	694
'Q^'= Highest Task Priority	682	'WK' Keep Window	695
'QD'= Windows Queue Display	682	'WL' Use Write Locks	695
'QF'= Task Priority Factor	683	'WP' Wait for Pipe on Close	695
'QK' Quick Key Lookup	683	'WT'= Number of Retries	695
'QS' START, Not Initialized	683	'WZ'= WindX ZLib Compression	696
'QT' No Prompt in Command Mode	683	'XC' WindX Continues After TCP Error	696
'RI' Round Multi-Line Inputs	683	'XF' Extended File Channels	696
'RN'= Rounding Control	684	'XI' Extract Ignore	696
'RP' Raw Print for *WINDEV*	684	'XL' Obsolete	696
'RR' Reset on RUN	684	'XS'= Extended Memory (KB)	697
'RS' Round STR()	684	'XT' ProvideX Exits to OS	697
'SB' Self-Block Extracts	686	'ZP' Accept Zero-Length Programs	697
'SC' Show Cursor	686	'!9' Sage MAS 90 Date Format	697
'SD' Subdirectory Slash	686	'!B'= Set Break Character	697
'SF' Short Form Variables	686	'!D' Numeric Separators: Legacy Mode	698
'SK' Shrink Keyed Files	687	'!F' Obsolete	698

'!' NOMADS Input Queue.....	698	'!U'= For Internal Use Only.....	699
'!K' Descending Key Logic (Legacy).....	698	'!V' I'm a Service.....	699
'!Q'= ODBC SQL Display.....	698	'!W' WindX Keyboard Synchronization.....	699
'!R'= For Internal Use Only.....	698	'!X' I/O Crossover.....	700
'!S' Suppress Error Flags on Serial Save.....	699	'*K' Obsolete.....	700
'!T' 'DP' or Decimal for Numerics.....	699	'*L' Obsolete.....	700

7. Control Object Properties

Overview.....	701	Properties List.....	709
Graphical Control Objects.....	703	Compound Properties.....	728

8. Special Files and Devices

Overview.....	737	*SYSTEM Event Handling Object.....	751
BITMAP Virtual Bitmap.....	738	*VIEWER* Print Preview.....	752
HTML Print to HTML.....	740	*WINDEV* Raw Print Mode.....	756
MEMORY Create & Use Memory File.....	741	*WINPRT* Windows Printing.....	760
PDF PDF Print Interface.....	744	*XML XML Interface.....	764

9. Special Command Tags

Overview.....	769	[OCI] Connect to Oracle Server.....	786
[DB2] DB2 Support.....	770	[ODB] Open DataBase.....	791
[DDE] Dynamic Data Exchange.....	776	[RPC] Remote Process Control.....	797
[DLL] Custom File Access.....	778	[TCP] Transmission Control Protocol.....	799
[LIB] Program Library.....	781	[WDX] Direct Action to Client Machine.....	801
[MYSQL] MySQL InnoDB Support.....	783		

A. Appendix

Overview.....	809
Input/Output and Control Options.....	810
Data Format Masks.....	813
Labels/Logical Statement References.....	816
Negative CTL Definitions.....	817
Operators.....	821
Apostrophe Operator.....	823
System Limits.....	825
Reserved Words.....	827
Error Codes and Messages.....	828
List of Messages.....	829

Index.....	843
------------	-----



Preface

The *ProvideX Language Reference* describes all features of the ProvideX programming language: directives, functions, system variables, mnemonics, parameters, specialty files, reserved words and system limitations. Although this volume is intended primarily for programmers and analysts, and describes the language in precise detail, it does not try to explain how to design and implement applications that can be written in ProvideX.

For product licensing and installation, refer to the *ProvideX Installation and Configuration Guide*. For a comprehensive look at the ProvideX environment, its uses, and the details required to develop applications, refer to the *ProvideX User's Guide*. Other ProvideX products (i.e., NOMADS, WindX, JavX, the ODBC Driver, the WebServer, and the Application Server) are fully documented in separate publications. Rather than reproduce existing material, references to these publications are supplied where applicable.

Using this Documentation

This documentation is designed for both viewing and printing via **Acrobat® Reader**. Click **Help > Reader Guide** on the menu bar to learn how to display, copy, search, and print PDF documentation. While there are several ways to navigate the contents of a PDF-based document, the following methods are highly effective, and are consistent with other documentation distributed by ProvideX:

Bookmarks

The list of bookmarks, displayed on the *left side* of the Acrobat window, serves as a hyperlinked *table of contents*. Bookmarks are displayed in a hierarchy where subordinate headings appear indented below main headings. When subordinates are hidden or *collapsed*, a plus sign (in Windows) or triangle (in Mac OS) will appear next to the main heading. Simply click on the *plus sign* or *triangle* to display all collapsed headings.

Cross-References

Blue hyperlinks appear throughout this document wherever one section cross-references another. They also appear in the form of hyperlists; such as the *Table of Contents*, the *Index*, and the linked tables placed at the beginning of some chapters.

The mouse pointer looks like an index finger when it is positioned over a linked cross-reference — simply click to activate the link. For example, "[Using this Documentation](#)" is hyperlinked back to the beginning of this section.



PDF Navigation Tips: The *chapter name* at the top left corner of the page head can be used as a hyperlink to the beginning of the chapter. Use the page-up/down/back/forward buttons ▼ ▲ ◀ ▶ to move one page at a time. Use the *Back* button ◀◀ to jump to the *previous view*.

Conventions

The following syntax items are used in this documentation to illustrate the format of program statements in ProvideX.

- ... Dots indicate the continuation of a list of elements.
- [] Square brackets enclose optional elements in the format. For example, in **ABS(num[,ERR=stmtref])** you can omit the **ERR=stmtref** portion of the statement as in **ABS(X-Y)**. (Exceptions are noted for individual commands where the brackets are "real"; i.e., part of the syntax.)
- { } Curly brackets enclose a list of elements in syntax formats where it is mandatory to select one item. For example, with **{YES | NO}**, you must select either **YES** or **NO**. In descriptions in this manual, they denote {bitmap / icon} buttons. (Exceptions are noted for individual commands where the brackets are "real"; i.e., part of the syntax.)
- | Vertical bars (pipes) separate a choices; e.g., **{YES | NO}**.
- chan* Channel or logical file number. It must be an integer between 0 and 127. This identifies the channel to which your directive applies; e.g., **CLOSE (14)**.
 - Channel zero (0) is the console. If you omit the channel, the system defaults to 0 (the console).
 - Channels 1 to 63 are commonly used for local files.
 - Channels 64 to 127 are used for global files.
 - **Exception:** In extended file mode ('**XF**' system parameter) the channels range from 0–32767 for local files, and 32768–65000 for global files.
- col,ln, wth,ht* Position/coordinates. Numeric expressions. Column and line coordinates for top left corner, width in number of columns, and height in number of lines.
- ctrlopt, fileopt* Optional syntax elements — three-character codes followed by an equals sign and argument (DOM=3250).

stmtref Statement reference. This can be either the line label or line number of a statement in the current program. Line numbers must be in the range of 0 to 64999.

If your given line number does not exist, ProvideX goes to the statement with the next higher line number. For example, if line 1000 doesn't exist and 1010 is the next line number, then for GOTO 1000 ProvideX will go to 1010 and proceed with execution from there.

Exception: ProvideX verifies the existence of an IOLIST and *stmtref* for **IOL=stmtref**. It does not proceed to the next higher statement number.

varlist List of comma-separated variables. Typically, a mix of string and/or numeric variables is acceptable; e.g., DEPT , ITEM , DESC\$. . . (See individual directives for restrictions.)

Numeric Expressions and Variables

When a syntax format in this manual includes a *numeric* variable like *chan*, *index* or *num* (lowercase), you can normally substitute a *numeric expression* consisting of variables, literals, functions, and operators. For instance, your value could be something like HFN or 4 or NUM(A1\$) * 3 - 2. (NUM in upper case is the function.) When numeric variables are used in numeric expressions, subscripts are allowed; e.g., COST[4].

Example:

To apply the format **FOR var=first TO last[STEP val] ...**

```
FOR I=1 TO 10 or
FOR LEAPS=-10 TO XYZ STEP ABC*.1
```



Note: Exceptions and valid values are stated when there are restrictions on the use of numeric or string expressions in a format (e.g., where only variable names are allowed).

String Expressions and Variables

When a syntax format in this manual includes a *string* variable like *prog_name\$* or *title\$*, you can normally substitute a *string expression* consisting of variables, literals, functions, and operators; e.g., PRINT "Printing " + REPORT\$. When string variables are used in string expressions, subscripts and substrings are allowed; e.g., CUSTOMER\$(15 , 4).

Example:

For the **CHECK_BOX READ [*]ctl_id,state\$[,mode\$][,ERR=stmtref]** format, you need string variables to receive the current *state* and optionally, the *mode* of selection.

```
CHECK_BOX READ 14000 ,ON_OFF$,KEYSTROKE$
```

Chapter Outlines

Chapter 1. Introduction, p.17. Provides some general information about the ProvideX development system, introduces basic language concepts, and lists punctuation/syntax conventions.

Chapter 2. Directives, p.27. Provides an alphabetical listing of all ProvideX directives (commands), complete with mandatory and optional syntax elements.

Chapter 3. System Functions, p.389. Provides an alphabetical listing of the standard functions.

Chapter 4. System Variables, p.555. Provides an alphabetical listing of the various system variables used to provide system information such as the date and time.

Chapter 5. Mnemonics, p.577. Provides an alphabetical listing of the mnemonics used in ProvideX to control the output of information to terminals and printers.

Chapter 6. System Parameters, p.653. Provides an alphabetical listing of the system parameters that are normally used at start-up, to define a system's operation under ProvideX.

Chapter 7. Control Object Properties, p.701. Provides a list of control object properties and explains use of the apostrophe operator.

Chapter 8. Special Files and Devices, p.737. Describes ProvideX device files designed for special file handling.

Chapter 9. Special Command Tags, p.769. Describes the special command file tags that are used in conjunction with a pathname in an **OPEN** statement.

Appendix, p.809. Discusses additional features and contains information that is supplementary to this language reference.

Index, p.843. Contains a comprehensive list of keyword references. As with the *Table of Contents*, the page numbers in the Index are linked to the source.



1

Introduction

Welcome to ProvideX – a powerful, versatile, intuitive programming language and integrated development environment for building sophisticated business applications.

About ProvideX

At the most elementary level, ProvideX is a system that executes computer instructions written in the ProvideX language. The instructions may be entered one statement at a time for immediate execution, or contained in a program for sequential execution.

ProvideX also comprises, in itself, all the tools and facilities necessary to design, develop, and implement comprehensive multi-user applications – applications that are ready to accommodate industry-standard technologies and a variety of host platforms. This system is equipped with a suite of integrated development tools, automatic error correction, rigorous security controls, transparent access to commercially-available databases, support for the latest industry standards and protocols, and much more.

For Business Applications

Since the language is oriented towards integer math and values with two decimal points (monetary values), ProvideX is probably best known for the development of business-related products. It comes complete with a robust native data storage system that can handle generous file sizes and various file types. While optimized for small to mid-range businesses, ProvideX also has the capacity to manage large multi-user banking, manufacturing, and hotel/hospitality applications.

Unique Implementation

Programs created in ProvideX must be executed on systems where ProvideX is running. The fact that the development cycle and program execution occur within the same environment presents some distinct advantages in functionality: platform independence, unmediated debugging of source code, smaller program size, and the implementation of *Object-Oriented Programming* techniques.

This added functionality does not compromise performance. Unlike other similar languages, ProvideX uses a two-pass system that "pre-compiles" programs when they are saved to disk so they execute much faster at runtime.

Product Options

ProvideX can be configured for multiple uses (depending on the license and the platform), but every installation begins with the *base system* that includes:

- *ProvideX*. Language interpreter and application development environment.
- *NOMADS*. Toolset for developing GUI-based applications (*MS Windows*).
- *COM/OCX/ActiveX, DDE, DLL, ZLIB, SSL, PDF, etc.* Built-in support for a number of industry-standard technologies.

Extend the functionality of the ProvideX base system with a set of tightly integrated application development and deployment solutions:

- *WindX, JavX, and UltraFX*. Thin-clients for displaying and interacting with GUI-based ProvideX applications running from a server.
- *Application Server*. Secure configurable hosting facility for connecting thin-client implementations via MS Windows, UNIX/Linux, and MAC OS X.
- *Local and Client/Server ODBC*. Open DataBase Connectivity (ODBC) for external access to ProvideX databases.
- *Web Server*. Interface for producing ProvideX-coded websites that allow browser access to ProvideX and ODBC data sources.
- *Internet Toolkit*. Utilities for developing e-mail/web-enabled applications.
- *RPC. Remote Processing Capability* for distributed processing of ProvideX.
- *XML Support*. Implementation for parsing and serializing XML documents.
- *OCI, DB2, MySQL, ODBC*. Native external database support.
- *Smart Controls*. Auto-load capability for list boxes/grids.
- *Customizer*. Utility for customizing panels dynamically without changing source.
- *Multiple Image Support*. Extended support for a variety of image file types.
- *Report Writer*. Powerful interface for designing and generating reports (runtime module included with base system).
- *Views*. End-user "viewing" of application data for simplified extraction and reporting (runtime module included with base system).
- *Charting Control*. Control object for creating advanced chart illustrations.
- *OLE Server*. Interface allowing external applications to access ProvideX objects directly.



Note: These products may be sold as stand-alone add-on packages or as part of a *Professional* or *eCommerce* bundle. Contact your local ProvideX dealer/distributor or visit www.pvx.com for complete product information and licensing.

Basic Concepts

This section covers the structural concepts used throughout this manual. Some of the terminology discussed will be familiar to programmers with experience in other Business Basic languages. However, ProvideX has several unique properties. Before you attempt to program in ProvideX, take some time to understand the different aspects of the language.

For product licensing and installation, refer to the *ProvideX Installation and Configuration Guide*. For a comprehensive look at the ProvideX environment, its uses, and the details required to develop applications, refer to the *ProvideX User's Guide*.

ProvideX Session

The ProvideX environment comprises a *Command mode* and an *Execution mode*. When in Command mode, ProvideX will be waiting for a *directive* or *statement* to be entered. Execution mode begins once a **RUN** or a **CALL** directive is used to execute a program.

By default, ProvideX initializes in Command mode; as indicated by the Command mode prompt '->'.

Directives can be entered in Command mode. If a directive does not include a leading line number, it is executed immediately; otherwise, the statement is used in the construction of a program. When a statement is inserted into a program, the prompt changes from '->' to '-:!' to indicate that the program has been changed but not saved.

When ProvideX is in Execution mode, it receives and executes all the statements that constitute a program. The program remains in Execution mode until completed (via the **STOP** or **END** directive), an error occurs, or it is interrupted via a **BREAK** or **ESCAPE** instruction. The ProvideX session can be terminated using **BYE**, **QUIT**, or **RELEASE**.

These concepts are discussed in further detail in the *ProvideX User's Guide*.

Directives and Statements

In ProvideX, all processing is controlled by the use of *directives* – commands that tell the system what task is to be performed. Each program statement (line of code) consists of one or more directives. When ProvideX executes a program, it executes all directives contained in a statement from left to right, then proceeds to the next line. Some directives provide the ability to alter the normal flow of execution.

The general format of a program statement includes a unique line number (optional), the directive indicating the operation to perform, parameters, and comments. Parameters are syntax elements, keywords, operators, and arguments that can be used to further define a directive's operation. For example, the complete syntax for the **CLOSE** directive appears as follows: **CLOSE** (*chan* [, **ERR**=*stmtrf*]) [*chan* [, **ERR**=*stmtrf*)]...]. Depending on the statement, it is possible to exclude all but the mandatory parameters from the directive, as in **CLOSE** (14).

See [Chapter 2. Directives, p.27](#), for the complete list of directives and the details of their mandatory or optional parameters.

System Functions

A ProvideX *system function* consists of a three-character function name followed by an open parenthesis, the parameter(s) for the function, an optional error transfer, and finally a close parenthesis, e.g., `AND (A$, B$, ERR=0300)`. The number and type of parameters vary from function to function. All may include the **ERR=** option (even those where an error is unlikely). System functions are listed and described in [Chapter 3. System Functions, p.389](#).

Data and Variables

ProvideX supports two basic types of data -- *Numeric* data and *String* data. Numeric data consists of numeric values such as account balances, prices and quantities. String data consists of textual information such as account names and descriptions. A ProvideX program maintains and processes its data using *variables*: numeric variables to store numeric values and string variables to store textual information.

The language includes various internally-defined variables for providing system information, such as the date and time. These are listed and described in [Chapter 4. System Variables, p.555](#). You can use *system variables* wherever normal program variables would be used, but you cannot modify them.



Note: All system variables have reserved three-character names and do not have a trailing dollar sign \$, e.g., **CTL**. To avoid potential conflicts with the reserved list (since ProvideX might reserve more three-character variables in the future), Sage Software Canada Ltd. recommends strongly that you do not use three-character variable names.

Mnemonics

Mnemonics are used to control output to terminals and printers. A mnemonic instruction is enclosed in single quotation marks. Refer to [Chapter 5. Mnemonics, p.577](#), for an alphabetical listing of all ProvideX mnemonics, complete with formats and descriptions. Refer to the [MNEMONIC Directive, p.210](#) for more information.

Examples:

```
5000 PRINT @(5,5), 'CL' ! Clears screen-line 5 to its end, starting at column 5
5150 OPEN (30) PRINTER$
5160 PRINT (30)'FF', ! Form-feed instruction to PRINTER$ on Channel 30
```

System Parameters

System parameters are normally used at start-up to define the system's operation under ProvideX. For example, the **BY** parameter is used to define the base year for the **JUL()** and **DTE()** functions. Like mnemonics, system parameters are enclosed in single quotation marks. Refer to [Chapter 6. System Parameters, p.653](#), for an alphabetical listing of system parameters, complete with formats and descriptions. For further information, refer to the [PRM\(\) Function, p.504](#), and the [PRM System Variable, p.570](#).

Graphical Control Objects

Graphical control objects in ProvideX programs display information, input data, and handle event processing. The following directives are used to create and maintain the various control object types: **BUTTON**, **CHART**, **CHECK_BOX**, **DROP_BOX**, **GRID**, **LIST_BOX**, **MULTI_LINE**, **RADIO_BUTTON**, **TRISTATE_BOX**, **VARDROP_BOX**, **VARLIST_BOX**, **V_SCROLLBAR**, and **H_SCROLLBAR**.



Note: In Windows, the above directives use Graphical Device Interface (GDI) resources/handles that are only released when the window they are in is dropped or cleared.

Graphical control objects can also be produced using the NOMADS toolset for GUI-based application development. Refer to the *ProvideX NOMADS Reference*. The attributes of a control object can be referenced and determined by the use of *property names* ('Height', 'Font\$', 'Text\$', 'TextColour\$', ...). The object itself is defined by a numeric variable containing the CTL value associated with the control, followed by an apostrophe and the property. Refer to [Chapter 7. Control Object Properties, p.701](#), for a description of the [Apostrophe Operator](#), and to review all properties listed in the [Properties List](#).

Specialty Files

ProvideX supports the use of a series of *specialty files*. See [Chapter 8. Special Files and Devices, p.737](#) for descriptions of the commands you can use with the following special device files:

- ***BITMAP*** Virtual Bitmap, [p.738](#)
- ***HTML*** Print to HTML, [p.740](#)
- ***MEMORY*** Create & Use Memory File, [p.741](#)
- ***PDF*** PDF Print Interface, [p.744](#)
- ***VIEWER*** Print Preview, [p.752](#),
- ***WINDEV*** Raw Print Mode, [p.756](#),
- ***WINPRT*** Windows Printing, [p.760](#).

Special Command Tags

ProvideX supports the use of a series of *special file tags*. See [Chapter 9. Special Command Tags, p.769](#) for information on the following:

- [DDE] Dynamic Data Exchange, [p.776](#),
- [DLL] Custom File Access, [p.778](#),
- [LIB] Program Library, [p.781](#)
- [OCI] Connect to Oracle Server, [p.786](#),
- [ODB] Open DataBase, [p.791](#),
- [RPC] Remote Process Control, [p.797](#),
- [TCP] Transmission Control Protocol, [p.799](#),
- [WDX] Direct Action to Client Machine, [p.801](#).

Object Oriented Programming

The ProvideX language has been extended to support all of the key design principles of *Object Oriented Programming* (OOP). Various OOP-related articles and examples are available from the ProvideX website, www.pvx.com.

The three basic principles of OOP are *Encapsulation*, *Inheritance*, and *Polymorphism*. These concepts provide a major part of the framework used to create and interact with *objects*. ProvideX OOP objects are defined through the use of *classes*. A class defines the name given to the object definition, as well as the object's *properties* and *methods*. Properties are the data portion of the object, consisting of fields or variables. Methods (*or functions*) are the actions that the object will perform. The defined properties/methods are accessible via the **Apostrophe Operator**.

The following directives and functions are used to handle OOP mechanisms in ProvideX :

DEF CLASS , p.65	Defines the object class.
PROPERTY , p.261	Declares data/properties for the object.
LOCAL , p.197	Declares internal data/properties for the object.
FUNCTION , p.137	Declares functions or methods for the object.
LIKE , p.174	Specifies other objects that this object inherits from.
PROGRAM , p.259	Defines the default program that contains the object logic.
PRECISION , p.248	Sets default program precision for use within the object.
LOAD CLASS , p.195	Pre-loads a class definition into memory from a .pvc file.
RENAME CLASS , p.283	Change name of an existing class.
STATIC , p.329	Dynamically declares LOCAL variables.
DROP OBJECT , p.104	Deletes an object.
DROP CLASS , p.102	Deletes class definition and all related information.
NEW() , p.489	Creates an object instance.
REF() , p.512	Controls reference counts.

Accessing Data Files

ProvideX supports a wide variety of file formats: proprietary/non-proprietary, program, link, device, data, etc. When referring specifically to *data files* in ProvideX, the primary file types and record formats are defined as follows:

Serial	Records can vary in length and are typically accessed in a <i>sequential</i> manner from beginning to end.						
Indexed	Records are the same length and are accessed by index number.						
Keyed	Records are accessed via key, a string of characters used to identify the records in a file. Keyed files are the most common data file type used by applications written in ProvideX. Three Keyed formats are supported: FLR (fixed-length records), VLR (variable-length records), and EFF (enhanced file format). Keyed files are also considered to be: <table> <tr> <td>Direct</td> <td>consisting of a single key per record (FLR/VLR)</td> </tr> <tr> <td>Keyed</td> <td>consisting of one or more keys per record (FLR/VLR, EFF)</td> </tr> <tr> <td>Sort</td> <td>consisting of keys but no data (FLR/VLR, EFF).</td> </tr> </table>	Direct	consisting of a single key per record (FLR/VLR)	Keyed	consisting of one or more keys per record (FLR/VLR, EFF)	Sort	consisting of keys but no data (FLR/VLR, EFF).
Direct	consisting of a single key per record (FLR/VLR)						
Keyed	consisting of one or more keys per record (FLR/VLR, EFF)						
Sort	consisting of keys but no data (FLR/VLR, EFF).						

Access to a data file is controlled by the use of a *channel* or *logical file number*. The link between data file and channel is established using the **OPEN** directive. All subsequent input and output to the file uses the same channel, which cannot be reused until the file is closed (via **CLOSE**, **START**, **BEGIN**, or by termination of the user session). The directives listed below are used to create, delete, and rename data files:

CREATE TABLE Create Keyed File (EFF), p.58
DIRECT Create File with Keyed Access, p.89,
DIRECTORY Create Subdirectory, p.91
INDEXED Create Indexed File, p.159,
KEYED Create Single/Multi-Keyed File, p.166,
SERIAL Create a Sequential File, p.302,
SORT Create File for Sorting, p.327
RENAME Change a File's Name, p.282
ERASE Delete File/Directory from System, p.120
ADD INDEX Add Key to Keyed File, p.29
DROP INDEX Drop Key from Keyed File, p.103
RENAME..INDEX Rename Keys in Keyed File, p.285

The following directives are used for file input/output and access:

PRINT Display Information, p.255,
FIND Locate and Read Data, p.131
READ Read Data from File, p.271,
EXTRACT Read and Lock Data, p.126
REMOVE Delete Record from File, p.281,
PURGE Clear Data from a File, p.263
WRITE Add/Update Data in File, p.383
OPEN Open for Processing, p.232
CLOSE Close File, p.56
LOCK Reserve File for Exclusive Use, p.200
UNLOCK Remove Exclusive Use from File, p.349.

There are also several system functions for file processing:

FIB() Return File Information Block, p.434,
FID() Return File Information Descriptor, p.438,
FIN() Return File Information, p.441,
IND() Return Next Record Index, p.457,
KEC() Return Key of Current Record, p.465,
KEF() Return First Key of File, p.466,
KEL() Return Last Key of File, p.467,
KEN() Return Key After Next, p.468,
KEP() Return Prior Record's Key, p.469,
KEY() Return Key of Next Record, p.470,
KGN() Generate Record Key, p.471,
RCD() Return Next Record, p.508,
RNO() Return Next Record Number, p.514.

Refer to the *ProvideX User's Guide* for further information on handling data files.

Other Facets of ProvideX

The appendix at the end of this manual expands on several additional features of the ProvideX programming language. Subject sections are listed as follows:

Input/Output and Control Options, *p.810*

Data Format Masks, *p.813*

Labels/Logical Statement References, *p.816*

Negative CTL Definitions, *p.817*

Operators, *p.821*

System Limits, *p.825*

Reserved Words, *p.827*

Error Codes and Messages, *p.828*

Error Reporting and Handling

If an error condition is detected during the execution of a program, the associated error code(s) can be obtained using the **ERR** and **RET** system variables. The line number of the last system-detected error can be determined by the **ERS** variable. The **MSG()** function provides a description for any known ProvideX error in the range of 0 to 255. The system parameter '**ES**', if enabled, can be used to display any OS error messages, along with the normal ProvideX error, from a Command prompt.

By default, when an error occurs in an application, ProvideX will stop processing, display the error code/message and statement where it occurred, and then return to Command mode. Most ProvideX directives and functions, for which errors are anticipated, provide an error transfer option denoted by an **ERR=stmtref** option in the syntax description (*stmtref* represents a line number or label to transfer control to). Error handling can also be specified via the **SETERR** directive. The **ERROR_HANDLER** directive is used to assign a generic application-wide error handling program that will intercept any un-trapped errors in an application.

For further information on error codes and error handling in ProvideX, refer to the *ProvideX User's Guide*, as well as the following sections in this manual:

Error Codes and Messages, *p.828*

ERR System Variable, *p.560*,

ERS System Variable, *p.560*

RET System Variable, *p.571*,

MSG() Function, *p.484*,

'**ES**' System Parameter, *p.664*

SETERR Directive, *p.316*

ERROR_HANDLER Directive, *p.121*.

Punctuation/Syntax

The following syntax symbols have fixed meaning in ProvideX:

- ! **Exclamation.** ProvideX accepts an exclamation mark as a substitute for the **REM Remark**; e.g., ! this remark. An exclamation mark as the leading character of a string also denotes one of Sage Software Canada Ltd.'s embedded bitmaps; e.g., !STOP.
- " **Quotes.** Standard quotation marks enclose string literals. A leading quotation mark can also be used as a substitute for the **INVOKE** directive; e.g., "NOTEPAD is the same as INVOKE "NOTEPAD" .
- \$ **Dollar sign.** A dollar sign at the end of a variable name marks a string variable; e.g., CUST\$. Dollar signs can also enclose hexadecimal values, for example \$8A\$.
- ' **Apostrophe.** Single quotation marks (apostrophes) enclose system parameters and mnemonics, for instance 'TL' and 'CS'. The **Apostrophe Operator**, is used to indicate a control object property.
- ;
Semicolon. Directives and entry points are separated by semicolons in program statements. When entered as the first character of a line, ProvideX hides the line from line listings making it appear as if it did not exist. The line will execute correctly, but it cannot be interrogated.
- * **Asterisk.** ProvideX includes a number of auxiliary applications that are stored under the LIB directory. The names of these utilities and subsystems are preceded by an asterisk when accessed in ProvideX; e.g., *UPB, *IT. An asterisk may also have specific meaning in the syntax of different directives or functions; e.g., as a wildcard character.
- % **Percent Sign.** A percent sign *before* a variable name denotes a *global* variable or function; e.g., %DEPT. A percent sign *following* a variable name indicates that the variable is an *integer*; e.g., DEPT%. A variable name having both leading and trailing percent signs denotes a global variable for integer values; e.g., %DEPT%
- *[] **Asterisk + Square Brackets.** The *search utility* (for searching programs) is invoked by enclosing a search string within square brackets preceded by an asterisk.; e.g.,

```
->*[print]
0090 REM Printing
0100 PRINT DAY
0120 PRINT "Today's date is ",DAY
0610 IF LEN(X$)>100 THEN PRINT "TOO LONG"; GOTO 0210
```
- *[]=[] Global *search and replace* can be used to make changes in programs; e.g.,

```
*[CST$]=[CUST$]
```

changes all instances of CST\$ to CUST\$.

- : **Prompts.** When your ProvideX prompt is a dash with a colon, that indicates that your current program has not been saved. After you save your program, the prompt reverts to an arrow. Under WindX, the prompt is a dash and a right brace.
- >
- }
- / or \ **Slashes.** ProvideX accepts either slash (forward or back) as a substitute for **LIST**; e.g., / 30 is the same as LIST 30.
- xxxx: **String–trailing colon.** Use a trailing colon to denote that your string is a line label (statement reference or entry point); e.g.,


```
0110 IF UPDATE$="Y" GOSUB CUSTOMER
...
2000 CUSTOMER:
2010 INPUT 'CS',@(5,5),"Enter customer number",CST
2020 ! REST OF ROUTINE ...
2200 RETURN
```
- ? **Question Mark.** ProvideX accepts a leading question mark as a substitute for **PRINT**; e.g., ? CUST\$ is the same as PRINT CUST\$. ProvideX also places a question mark between a line number and program statement to denote a syntax error.
- ' **Back Apostrophe.** ProvideX accepts the back apostrophe as a substitute for the **EDIT**.



2

Directives

ACCEPT	DROP_BOX	INVOKE	PURGE	SETDRIVE
ADD INDEX	DROP_CLASS	IOLIST	QUIT	SETERR
ADDR	DROP INDEX	KEYED	RADIO_BUTTON	SETESC
AUTO	DROP OBJECT	KEYED LOAD	RANDOMIZE	SETFID
BEGIN	DROP.ON	LET	READ	SETMOUSE
BREAK	DUMP	LIKE	READ DATA	SETTIME
BUTTON	EDIT	LINE_SWITCH	READ RECORD	SETTRACE
BYE	ENABLE	LIST	REDIM	SHORT_FORM
CALL	ENABLE CONTROL	LIST_BOX	REFILE	SHOW_FORM
CASE	ENABLE EVENT	LOAD	RELEASE	SORT
CHART	END	LOAD CLASS	REM	START
CHECK_BOX	END DEF	LOAD DATA	REMOVE	STATIC
CLEAR	END SWITCH	LOCAL	RENAME	STOP
CLIP_BOARD	END WITH	LOCK	RENAME CLASS	SWITCH..
CLOSE	END IF	LONG_FORM	RENAME CONTROL	SYSTEM_HELP
CONTINUE	ENDTRACE	MENU_BAR	RENAME..INDEX	SYSTEM_JRNL
CREATE TABLE	ENTER	MERGE	RENUMBER	TABLE
CWDIR	ERASE	MESSAGE_LIB	REPEAT..	TRANSLATE
DATA	ERROR_HANDLER	MNEMONIC	RESET	TRISTATE_BOX
DAY_FORMAT	ESCAPE	MSGBOX	RESTORE	UNLOCK
DEF CLASS	EXECUTE	MULTI_LINE	RETRY	UNTIL
DEF GID/UID	EXIT	MULTI_MEDIA	RETURN	UPDATE
DEF FN	EXITTO	NEXT	ROUND	USER_LEX
DEF MSG	EXTRACT	NEXT RECORD	RUN	VARDROP_BOX
DEF OBJECT	EXTRACT RECORD	OBTAIN	SAVE	VARLIST_BOX
DEF systab=	FILE	ON EVENT	SAVE CONTROL	V_SCROLLBAR
DEF sysvar=	FIND	ON ... GOSUB	SAVE DATA	VIA
DEFAULT	FIND RECORD	ON ... GOTO	SAVE FILE	VIDEO_PALETTE
DEFCTL	FLOATING POINT	OPEN	SELECT..	WAIT
DEFPRT	FOR..	PASSWORD	SERIAL	WAIT FOR EVENT
DEFTTY	FUNCTION	PERFORM	SET_FOCUS	WEND
DELETE	GET_FILE_BOX	POP	SET_NBF	WHILE..
DELETE OBJECT	GOSUB	POPUP_MENU	SET_PARAM	WINPRT_SETUP
DICTIONARY	GOTO	PRECISION	SETCTL	WITH
DIM	GRID	PREFIX	SETDAY	WRITE
DIRECT	H_SCROLLBAR	PREINPUT	SETDEV	WRITE RECORD
DIRECTORY	HIDE	PRINT	SETDEV IOL=	
DISABLE	IF.. THEN.. ELSE..	PROCESS	SETDEV KEY	
DISABLE CONTROL	INDEXED	PROCESS SERVER	SETDEV PROGRAM	
DISABLE EVENT	INPUT	PROGRAM	SETDEV SEP=	
DROP	INSERT	PROPERTY	SETDEV TSK()	

Overview

This chapter provides an alphabetically arranged list of all ProvideX directives. Each definition includes the correct syntax (showing associated parameters), values returned, a general description, examples, and sometimes a cross reference to related material. The list begins on the following page.

ACCEPT Directive*Read Single Keystroke*

Format **ACCEPT**(*chan*[,*fileopt*])*char*\$

Where:

chan Channel or logical file number.

char\$ String variable. Receives the input character.

fileopt Supported file options (see also, [File Options, p.810](#)):

ERR=*stmtref* On error , transfer to program line number/label.

NUL=*stmtref* On no input, transfer to program line number/label.

TIM=*num* Maximum time-out value in integer seconds.

Description Use the **ACCEPT** directive to read a single keystroke (character) from a terminal. The **ACCEPT** directive doesn't echo the keystroke/character to the terminal. If no input is in the input buffer, the **ACCEPT** directive waits for terminal input. Control transfers to the *stmtref* if you include a **NUL=** option and the terminal input buffer has no data in it.

See Also [OBTAIN Get Hidden Terminal Input, p.227](#)
[INPUT Get Input from Terminal, p.160](#)

Example The following example reads one record at a time and displays it on the terminal. When any key is pressed, control drops to line 130 (ending the listing of the file).

```
0100 READ RECORD (1,END=0130)R$
0110 PRINT R$
0120 ACCEPT (0,NUL=0100,TIM=0.5)X$
0130 PRINT "<Display halted>"
```

ADD INDEX Directive

Add Key to Keyed File

Format	ADD INDEX <i>keydescription</i> TO <i>filename</i> \$ [, ERR = <i>stmtref</i>]
	<i>Where:</i>
<i>filename</i> \$	Name of the file to which the key will be added. String expression.
<i>keydescription</i> \$	Description of the key using the same format as KEYED directive.
<i>stmtref</i>	Program line number or label to transfer control to.
TO	Mandatory keyword, not case-sensitive.

Description The **ADD INDEX** directive allows keys to be added to a ProvideX Keyed file without having to rebuild the file. When adding keys to the file:

- ProvideX builds keys on-the-fly.
- Exclusive access to the file is required.
- Keys are assigned the next available key number.
- When adding a unique alternate key, the add generates an error and the key is not defined if duplicate keys are in the desired index.
- Only one key can be added at any one time (**ADD** or **DROP** cannot execute at the same time and will fail with an error zero).
- The key names are case insensitive (key will be converted to uppercase characters).
- The first character of the key name cannot be a "#" as this is reserved for access to the keys by key number.
- If a key name is specified, it cannot be null; e.g., "".
- Spaces are valid and significant. A name of " " (space) is valid, and is not the same as a key of " " (space space).



Note: A total key length that exceeds 240 characters will result in Error #80: Invalid key definition.

Example `ADD INDEX ["KeyName":2:1:30]+[1:1:6] TO "cstfile"`

See also [DROP INDEX Drop Key from Keyed File, p.103](#)
[RENAME..INDEX Rename Keys in Keyed File, p.285](#)

ADDR Directive *Load & Lock Program in Memory*

Format **ADDR** *prog_name*[\$[, **ERR**=*stmtref*]

Where:

prog_name\$ Name of the program to be loaded and kept in memory. String expression. If you omit the pathname, ProvideX search rules apply.

stmtref Program line number or label to transfer control to.

Description The **ADDR** directive loads the specified program into memory and keeps it there until you unload it.

For as long as the **ADDR**-loaded program is in memory, that specific program will be referenced (using its absolute path) even if you change the system **PREFIX** or current directory after executing the **ADDR** directive. That is, all subsequent references to any program whose name matches that string will still refer to the **ADDR**-loaded program.

You can see this in the example below. When the program is executed, ProvideX **ADDR**-loads "DATECHK", changes directories and executes the **ADDR**-loaded program even though there is a different current directory and a different program with the same name might exist in that directory.

To unload a program which is **ADDR**-loaded, either use the **DROP** directive or execute the **START** directive.

See also [DROP Removes Program from Memory, p.95](#)
[START Restart ProvideX, p.328](#)

Examples

```
0010 CWDIR "/opt/application/utilities"
0015 ! /opt/application/utilities/datecheck is loaded in memory.
0020 ADDR "datecheck"
0025 ! New working directory: could contain a different datecheck.
0030 CWDIR "/opt/application/ar"
0040 RUN "datecheck" ! Still runs /opt/application/utilities/datecheck
```

AUTO Directive

Automatic Line Generation

Format **AUTO** [*lineno*[,*incr*]]

Where:

incr Increment you want ProvideX to add automatically to generate each subsequent line number. Default increment is 10; however, you can set a different default via the '**A**'= [System Parameter, p.656](#).

lineno Starting program line number you want ProvideX to use. Optional.

Description Use the **AUTO** directive to have ProvideX automatically generate line numbers to prefix your statement input.



Note: This directive only works in Command mode.

If you omit the starting line number in this directive, then ProvideX uses either the last generated line number (if any), or the next higher line in the current program. If you do not specify an increment, the default is 10. If you change the increment in Command mode, it stays at the new setting until you change it again.

The **AUTO** function remains active until you enter a null line (e.g., press **Enter**) or execute a command. You can backspace over and change the generated line number.

Examples

```
->AUTO 100,10
0100 ! ProvideX generates lineno 0100 for you. Add the statement.
0110 ! ProvideX adds 10, generates line 0110.
0120
```

BEGIN Directive

Reset Files and Variables

Format **BEGIN** [[**EXCEPT**]*varlist*]

Where:

varlist List of variables. Optional.

Description The **BEGIN** directive performs the following functions:

1. Closes all local files currently open (*global files are not affected*).
2. Closes all open windows (unless the '**WK**' parameter is set), drops non-global control objects, and resets the menu bar.
3. Clears local variables (*global variables are not affected*).
 - a) All if no *varlist* appears on directive.
 - b) Only those in *varlist* if no **EXCEPT** clause given.
 - c) All but those in *varlist* if **EXCEPT** clause present.

Refer to the examples provided below.
4. Resets **PRECISION** to the default ('**PD**'=2) or value set via the '**PD**'= System Parameter, p.679.
5. Sets **ERR**, **RET**, and **CTL** to zero.
6. Clears the stack for **FOR..NEXT**, **WHILE..WEND**, **GOSUB..RETURN**, etc.
7. Resets pointer to the first **DATA** item in the program.



Note: When executed within an object (in *Object Oriented Programming*), the **BEGIN** directive will clear all of the object's properties, along with standard variables, and will close any standard local files and files owned by the object.

See Also

[CLEAR Reset Variables, p.54](#),
[RESET Reset Program State, p.288](#),
[START Restart ProvideX, p.328](#),
[ERR\(\) Test Error Value, p.427](#),
[CTL Control Signal, p.557](#),
[RET Operating System's Last Error Code, p.571](#).
[Data Integration, User's Guide](#).

Examples

The following examples refer to item 3 in the description.

- a) 0100 BEGIN
- b) 0100 BEGIN A3\$, B3\$, C3, D3
- c) 0100 BEGIN EXCEPT CST_ID\$, TX_VAL, TX_TBL\$ {ALL}

BREAK Directive

Immediate Exit of Loop

Format **BREAK**

Description Use the **BREAK** directive to perform an immediate exit from a loop or case structure. Execution resumes following the directive that would normally have ended the loop or case structure (e.g., **NEXT**, or **WEND**). The *BREAK label emulates a **BREAK** directive for use as a statement reference.



Note: You can now use **BREAK** commands in **SELECT** structures. In earlier version of ProvideX, **BREAK** and the corresponding *BREAK label were not supported for use with **SELECT..NEXT RECORD** directives.

See Also [CASE Define Branch Points, p.42](#)
[SWITCH..CASE Branch Control, p.331](#)
[EXITTO End Loop, Transfer Control, p.125](#)
Flow Overrides in the *ProvideX User's Guide*

Examples

```
00100 PROCESS_TAXCODE:
00110  LiquorTax=0,SalesTax=0,ServiceTax=0
00120  SWITCH UCS(TaxCode$)
00130  CASE "X","Z" ! two codes are tax exempt
00140  BREAK ! stop processing for case "X" here
00150  CASE "L" ! liquor pays all liquor,sales and service tax
00160  LiquorTax=cost*LiquorTaxRate
00170  ! no break here, logic falls through
00180  CASE "S" ! pays sales and service tax
00190  SalesTax=cost*SalesTaxRate
00200  ! no break here, logic falls through
00210  CASE "V" ! service tax
00220  ServiceTax=cost*ServiceTaxRate
00230  BREAK ! end processing for this case and any that fell through
00240  DEFAULT ! enter here if case not found
00250  MSGBOX "Unknown tax code","Error"
00260  END SWITCH
00270  TotalTax=LiquorTax+SalesTax+ServiceTax
00280  RETURN
```

BUTTON Directive

Control Button

Formats

1. *Define/Create*: **BUTTON** [*]ctl_id,@(col,ln,wth,ht)=contents\$[,ctrlopt]
2. *Remove*: **BUTTON REMOVE** [*]ctl_id[,ERR=stmtref]
3. *Disable/Enable*: **BUTTON** {DISABLE | ENABLE} [*]ctl_id[,ERR=stmtref]
4. *Hide/Show*: **BUTTON** {HIDE | SHOW} [*]ctl_id[,ERR=stmtref]
5. *Force Focus*: **BUTTON GOTO** [*]ctl_id[,ERR=stmtref]
6. *Signal on Focus*: **BUTTON SET_FOCUS** [*]ctl_id,ctl_val[,ERR=stmtref]
7. *Logical Push, Release*: **BUTTON** {ON | OFF} [*]ctl_id[,ERR=stmtref]
8. *Read Activation Mode*: **BUTTON READ** [*]ctl_id,mode\$[,ERR=stmtref]

Where

- * Optional. Use a leading asterisk to denote a *global* button.
- @(col,ln,wth,ht) Position and size of the button region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines. Use line value -1 to display the button on the tool bar.
- contents\$ Text/images to appear on the button. See **BUTTON contents\$, p.36**
- ctl_id Unique logical identifier for the button (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the **F4** key) or **Negative CTL Definitions, p.817**. Use this value with the apostrophe operator to access various **Button Properties**.
- ctrlopt Control options. Supported options for **BUTTON** include:
 - ERR=stmtref** Error transfer
 - FNT="font,size[,attr]"** Font name, size, optional properties
Refer to the **'FONT' Mnemonic, p.609** for details.
 - MSG=text\$** Message string.
 - MNU=ctl** CTL value associated with right-click menu event
 - OWN=name\$** Name assigned for automated testing of this control
 - OPT=char\$** (See **BUTTON OPT= Settings**)
 - TIP=text\$** Mouse pointer message
To change the colour, refer to the **'TC'= System Parameter, p.689**.
- ctl_val CTL value to generate when focus goes to button.
- mode\$ String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to activate the button (\$01\$ for **MOUSE-CLICK** or \$0D\$ for **Enter**).
- stmtref Program line number or label to transfer control to.

BUTTON OPT= Settings

Available attribute/behaviour settings are listed below. Some characters may be combined. Invalid settings are ignored.

"<" *Bitmap Left*. Places bitmap left of text.

- ">" *Bitmap Right*. Places bitmap right of text.
- "_" *Bitmap Below*. Places bitmap below text (4D only). See '**4D**' Mnemonic, p.586.
- "~" *Bitmap Above*. Places bitmap above text (4D only). See '**4D**' Mnemonic, p.586.
- "^" *Drop-down*. Adds drop-down functionality. (This requires menu event setting via **MNU=ctl** option or **MenuCtl** property.)
- "*" *Default*. Defines button as default.
- "B" *Bitmap Button*. Has a bitmap whose width is divided into four images. Use this attribute to custom design buttons of any colour, style or shape by controlling the bitmap image that appears. Each of the four divisions represents what a button will look like in a particular state:
 - 1st quarter: Bitmap image when button is disabled.
 - 2nd quarter: Bitmap image when button is in normal (released) state.
 - 3rd quarter: Bitmap image when the mouse is over the button.
 - 4th quarter: Bitmap image when the button is pressed.
- "d" *Permanently Disabled*. Button is grayed out and cannot be enabled.
- "D" *Initially Disabled*. Button is initially grayed out.
- "F" *Flat*. Button shows no raised outline unless the mouse is over the button or the button is pushed.
- "f" *Flat-No Border*. Same as "F", but has no border.
- "G" *Global*. Keep active when focus changes to new/non-concurrent window. When using secondary commands (**REMOVE** or **SET_FOCUS**) on buttons created with **OPT="G"** identify the button by prefixing the CTL value with an asterisk; e.g.,
`BUTTON 100,@(10,10,10,1)="Global",OPT="G"`
`BUTTON REMOVE *100`
- "h" *Permanently Hidden*. Button cannot be shown.
- "H" *Initially Hidden*. Button is initially hidden.
- "S" *Signal Only*. ProvideX generates a CTL value, but does not shift focus to the button automatically (the default), but only when focus is explicitly passed to it. Use this to have a button act like a function key.
- "s" *Scroll*. Button can scroll within a resizable/scrollable dialog box.
- "T" *Transparent*. Button is "see-through" to window contents below button area.
- "U" *Underscore*. Text is underlined.
- "V" *Hovertext*. Indicates that text will change color when mouse is over the button.
- "Y" *System Tray*. Places an icon in the *Taskbar Notification Area*.

Combined options can be used to create several different button types. For example, the "f", "T", and "U" options provide the ability to turn buttons into *hotspots*, which allows for clickable areas on bitmaps or hyperlinked text in dialogues; e.g.,

- "VTf" Creates a general hotspot.
- "VUTf" Creates an HTML-like hotspot (e.g., URL hyperlink).
- "F^" Creates a word-style toolbar with drop list

Description

Use the **BUTTON** directive to create/control a button object on the screen or to place an icon in the **Taskbar Notification Icon** (see *User's Guide*, p.155). The *ctl_id* is used to generate a CTL whenever the button is pressed. If *ctl_id* is prefixed by an *asterisk* *, the button is considered global, and not tied to a specific window. By default, non-global buttons are deleted when a window is removed/dropped or when the application issues a **BEGIN**. Global buttons can be removed manually or cleared by a **START**.

BUTTON contents\$

The *contents\$* string expression defines the text or picture to appear on the button. In the text, you can use an ampersand "&" preceding a character to identify it as a hot key the user can press in conjunction with the **Alt** key to activate the button from the keyboard.

Images and Icons

When adding an image to a button, enclose the image name in curly braces. Use a leading exclamation point (!) to identify the image as internal, or specify the relative path and filename to access an image file that is external. There are no icons in the ProvideX executable and ProvideX does not support retrieving icons from either resource libraries or other system DLLs /executables. For more information on the options available for displaying internal/external images and the recognized image file types, see **Images and Icons**, p.153 in the *User's Guide*.

When you use text as well as images, the relative positions of the image and the text set their relative placement. The following are example *contents\$* expressions:

```
"{!Add}Add" ! Displays the {!Add} bitmap in front of the text "Add"
"Delete{!Del}" ! Displays the {!Del} bitmap after the text "Delete"
```

If the string expression includes two images separated by a vertical bar inside a single set of curly braces, the first is displayed when the button is released (normal state), the second when the button is pressed; e.g., "{!Stop|C:\MYBMP\Go}.

You can also use the `OPT="B"` clause for a *Bitmap Button* to display different images for different states. See **BUTTON OPT= Settings**, p.34.

Button Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a button object are described in **Chapter 7. Control Object Properties**, p.703.

Format 1: *Define/Create*

```
BUTTON [*]ctl_id,@(col,ln,wth,ht)=contents$[,ctrlopt]
```

Use this format to create a button control object, and give it a unique identifier in *ctl_id*. The *ctl_id* is used to generate a CTL value whenever the user presses the button; e.g.,

```
0010 REM Create button: CTL=4000 when pressed, 'Stop Sign' picture, "Exit" text
0020 BUTTON 4000,@(2,14,12,2)="{!Stop}Exit"
```

If the `ctl_id` has a leading *asterisk* `*`, the button is considered global (not tied to a specific window); e.g.,

```
0010 REM To add a global button to the Toolbar with the text "Help":
0020 BUTTON *1000,@(0,-1,10,1)="Help"
```



Note: A button object can be created using a `ctl_id` of 0 zero for applications that need to emulate the **Enter** key in the absence of an actual keyboard; i.e., *touchscreen* format.

Format 2: *Remove*

BUTTON REMOVE [*]`ctl_id`[,**ERR=stmtref**]

Use **BUTTON REMOVE** to delete a button; e.g., to remove the button created previously,

```
0030 BUTTON REMOVE 4000
```

By default, non-global buttons are deleted when a window is removed/dropped or when the application issues a **BEGIN**. Global buttons can be removed using **BUTTON REMOVE** syntax, or cleared using the **START** directive.

Format 3: *Disable/Enable*

BUTTON {DISABLE | ENABLE} [*]`ctl_id`[,**ERR=stmtref**]

Use the **BUTTON DISABLE** format to gray out a button so that it will be visible but *inaccessible* to users. To reactivate it, use **BUTTON ENABLE**.

```
0030 BUTTON DISABLE 4000
...
0500 BUTTON ENABLE 4000
```

Format 4: *Hide/Show*

BUTTON {HIDE | SHOW} [*]`ctl_id`[,**ERR=stmtref**]

Use the **BUTTON HIDE** format to hide the display of an active button. (The user can't see or use the button when it's hidden. Use the **BUTTON SHOW** format to restore the button's display and access.

Format 5: *Force Focus*

BUTTON GOTO [*]`ctl_id`[,**ERR=stmtref**]

Use **BUTTON GOTO** to reactivate and force focus to a button, ready for the next user action; e.g.,

```
0030 BUTTON GOTO 4000
```

Format 6: *Signal on Focus*

BUTTON SET_FOCUS [*]`ctl_id`,`ctl_val`[,**ERR=stmtref**]

The **BUTTON SET_FOCUS** format can be used to define an alternate CTL value to generate whenever focus shifts to the button.

Format 7: Logical Push, Release**BUTTON {ON | OFF} [*]ctl_id[,ERR=stmtref]**

Use **BUTTON ON/OFF** to make it look like the button was pressed/released though no signal is generated; e.g.,

```
0030 BUTTON ON 4000
```

Format 8: Read Activation Mode**BUTTON READ [*]ctl_id,mode\$[,ERR=stmtref]**

Use **BUTTON READ** to receive the user's method of selecting the button. ProvideX returns a hex value in the *mode\$* variable to tell you what keystroke last activated the button. Once read, this field is reset to \$00\$. Possible values are described below:

\$01\$ for **MOUSE-CLICK**.

\$02\$ for **DOUBLE MOUSE-CLICK**.

\$0D\$ for **Enter**.

\$20\$ for **SPACEBAR** (and keyboard hot key, as in the example below).

\$09\$ for **Tab** to go to the button.

\$00\$ when the user exits the button control.

Other specific characters include: \$81\$ for **F1** \$82\$ for **F2** ...

Read the user's selection(s) to make them available to your applications. In the example below, the *STROKE\$* variable returns the user's method of selection (by mouse, carriage return or space bar). The value returned for keyboard strokes (the hot key **Alt-B** and the **SPACEBAR** for the following example, is \$20\$).

```
0010 ! BUTTON Example
0020 PRINT 'CS'
0030 LIST
0040 BUTTON 2000,@(24,17,7,3)="{!BUG|!STOP}Hit the &Bug"
0050 LET BTN=2000
0060 SETCTL BTN:READ_BOX
0070 BUTTON GOTO BTN
0080 PRINT @(24,24),"HotKey=<Alt-B>. Try Mouse and keyboard too. END=<F4>"
0090 OBTAIN (0,SIZ=1,ERR=0090)@(0,0),'CURSOR'("off'),'ME',IN_VAR$,'MN'
0100 IF CTL=4 THEN GOTO END
0110 READ_BOX:
0120 BUTTON READ BTN,STROKE$
0130 PRINT @(24,20),"Your HTA(selection)=" ,HTA(STROKE$) ," ,CTL=" ,CTL:"#####"
0150 GOTO 0090
0160 END:
0170 BUTTON REMOVE BTN; PRINT 'CS'
```

See Also

RADIO_BUTTON Control Radio Button, p.265

CHECK_BOX Control Check Box, p.47

TRISTATE_BOX Control Tristate Box, p.344

Chapter 7. Control Object Properties, p.701.

Images and Icons, p.153 in the User's Guide.

BYE Directive

Terminate ProvideX Session

Format **BYE**

Description Use the **BYE** directive to end a ProvideX session and return to the operating system. If the **ERR** system variable has a value other than zero (i.e., an error has occurred) then the operating system is informed that an error has occurred within ProvideX. This allows you to do external testing of error conditions.

The ProvideX compiler converts the **BYE** directive into a **QUIT** directive. When you use it in a compound statement, the **BYE** directive must be the final directive.

See Also [QUIT Terminate ProvideX, p.264](#)
[RELEASE Terminate ProvideX, p.279](#)

CALL Directive

Transfer to Subprogram

Format `CALL subprog$[:entry$][,ERR=stmtref][, arglist]`

Where:

;entry\$ Name of starting line label to use as entry point in the subprogram. Optional. Max string size 8kb. If included, use a leading semicolon and add it to the *subprog* string expression; e.g.,
`CALL "get_pizza;order", top_1, top_2 ...`

arglist Comma-separated list of variables, literals, or expressions.

stmtref Program line number or label to transfer control to.

subprog\$ Name of the subprogram to call. Max string size 8kb.

Description

Use the **CALL** directive to transfer control to a subprogram. The current program state is saved and the specified subprogram is loaded and executed. If you use arguments, they are recieved in the called subprogram via the **ENTER** directive.

The called program should terminate with an **EXIT**, which may be replaced by an **END** or **STOP**.

Arguments for CALL and ENTER

Normally, the total number of arguments in the **CALL** and **ENTER** statements must match. Each argument in the **CALL** statement must correspond in relative position and in type (numeric or string) to a variable in the **ENTER** statement. If you use a shorter list of arguments in a **CALL** statement than in the **ENTER** statement, make sure to maintain relative position and type up to the point where you shorten the list (and include error handling options). Otherwise, ProvideX returns

Error #36: ENTER parameters don't match those of the CALL.



Warning: If you pass an argument to the subprogram using a simple variable (e.g., *A\$, Z*) then any changes to the variable in the subprogram will have an effect in the calling program. Subscripted variables/expressions (including substrings) or any values enclosed in parentheses only have their values passed one way: to the subprogram. Changes made to these will *not* affect the calling program.

You can protect a simple variable in a **CALL** or **ENTER** statement by placing it inside parentheses. This turns the variable into an expression, which has the effect of making it read only; e.g., `CALL "PROG" , (A$)`.

IOLists can also be used as arguments for **CALL** statements; e.g.,

```
CALL "PROG" , IOL=8000
```


Pass a complete array to a subprogram by specifying the array name followed by {ALL}. In this case, all values are passed to the subprogram and any changes made are returned to the calling program. See the *ProvideX User's Guide* for more details on this directive.

String templates cannot be passed if they are defined prior to the **ENTER** statement in the called program.

CALL Using Entry-Point Labels

This directive also has an optional ProvideX feature you'll find useful for applications like subprogram "libraries" (with multiple stand-alone routines, each accessed by a line label). To use this form of access, append a semicolon plus the label name of the starting statement to the subprogram name (e.g., `CALL "PROG;STARTING_LABEL",ERR=1000,X$,A,CT$`). After the called subprogram is loaded, ProvideX internally issues a `GOTO STARTING_LABEL` (e.g.) directive and starts execution there. Use this to create a single subprogram with multiple entry points and **ENTER** directives.

CALL and ENTER from ASCII Programs

When you run programs from ASCII text files, the **CALL** and **ENTER** with no arguments will always fail because the system thinks that the variables have already been assigned. To work around this, use the **PERFORM** directive or save the called program in a program file.

See Also

Called Procedures, User's Guide
ENTER Specify Arguments, p.119
PERFORM Call Subprogram, Pass Variables, p.243
END Halt Program Execution, p.113
EXIT Terminate Subprogram and Return, p.124
STOP Halt Program Execution, p.330

Examples

```
0020 CALL "ABCDEF",ERR=0050,A,4*F,Z$,X$(4,5),(Q)
```

In ABCDEF:

```
0030 ENTER Z,X,A$,B$,V$
```

Z will receive the value of A – A will reflect changes in Z.

X will receive the value of 4*F.

A\$ will receive the value of Z\$ – Z\$ will reflect changes in A\$.

B\$ will receive the string from X\$(4,5) which will not change.

V\$ will receive Q\$ but Q\$ cannot be changed.

CASE Directive

Define Branch Points

Format **CASE** range[\$]

Where:

range[\$] List of values defining branch points in a program. String or numeric expression.

Description Use the **CASE** directive to list possible branch points in a program.



Note: Refer to **SWITCH..CASE Branch Control, p.331**, for complete syntax.

See Also **SWITCH..CASE Branch Control, p.331**,
BREAK Immediate Exit of Loop, p.33,
DEFAULT Branch If No Matching Case, p.77

Examples

```

00100 PROCESS_TAXCODE:
00110  LiquorTax=0,SalesTax=0,ServiceTax=0
00120  SWITCH UCS(TaxCode$)
00130  CASE "X","Z" ! two codes are tax exempt
00140  BREAK ! stop processing for case "X" here
00150  CASE "L" ! liquor pays all liquor,sales and service tax
00160  LiquorTax=cost*LiquorTaxRate
00170  ! no break here, logic falls through
00180  CASE "S" ! pays sales and service tax
00190  SalesTax=cost*SalesTaxRate
00200  ! no break here, logic falls through
00210  CASE "V" ! service tax
00220  ServiceTax=cost*ServiceTaxRate
00230  BREAK ! end processing for this case and any that fell through
00240  DEFAULT ! enter here if case not found
00250  MSGBOX "Unknown tax code","Error"
00260  END SWITCH
00270  TotalTax=LiquorTax+SalesTax+ServiceTax
00280  RETURN

```


CHART Directive

Control Chart

Formats

1. *Define/Create*: **CHART** *ctl_id*,@(col,ln,wth,ht),[,ctrlopt]
2. *Remove*: **CHART REMOVE** *ctl_id*[,ERR=stmtref]
3. *Disable/Enable*: **CHART** {**DISABLE** | **ENABLE**} *ctl_id*[,ERR=stmtref]
4. *Hide/Show*: **CHART** {**HIDE** | **SHOW**} *ctl_id*[,ERR=stmtref]
5. *Load*: **CHART LOAD** *ctl_id*,strvar\$[,ERR=stmtref]
6. *Clear Data*: **CHART CLEAR** *ctl_id*[,ERR=stmtref]
7. *Clear Data & Titles*: **CHART DELETE** *ctl_id*[,ERR=stmtref]
8. *Retrieve Value*: **CHART FIND** *ctl_id*,dataset,point,{numvar|label\$}[,ERR=stmtref]
9. *Read Selected Set*: **CHART READ** *ctl_id*, dataset,eom\$[,ERR=stmtref]
10. *Update Existing Values*: **CHART WRITE** *ctl_id*,dataset,point,{numvar|label\$}[,ERR=stmtref]

Where

@(col,ln,wth,ht)	Position and size of the check box region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.
ctl_id	Unique logical identifier for the chart (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the  key) or Negative CTL Definitions, p.817 . Use this value with the apostrophe operator to access various Chart Properties .
ctrlopt	Control options. Supported options for CHART include: ERR=stmtref Error transfer FMT=def\$ See <i>Chart Formats</i> listed below.) FNT="font" Font name (size controlled by window coordinates). OPT=char\$ (See <i>Attribute/Behaviour Settings</i> below). SEP=char\$ Delimiter character. Hex or ASCII string value. TIP=text\$ Mouse pointer message. To change the colour, refer to the 'TC'= System Parameter, p.689 . <i>Chart Formats</i> (for FMT=def\$): 2DLINE (<i>default</i>), 2DAREA, 3DAREA, 2DBAR, 3DBAR, 2DCOLUMN, 3DCOLUMN, 3DLINE, 2DPIE, 3DPIE, 2DRIBBON, 3DRIBBON, 2DSCATTER, 3DSCATTER, 2DSTACK, 3DSTACK . <i>Attribute/Behaviour Settings</i> (for OPT=char\$): " B " - Chart has no border or frame. " D " - Initially disabled " d " - Permanently disabled " H " - Chart is initially hidden " h " - Permanently hidden. " G " - Keep active on focus change to a new / non-concurrent window. Some OPT= characters may be combined. Invalid settings are ignored.
dataset	Dataset from which values will be drawn.
numvar	Name of variable that will contain returned numeric data value.
eom\$	EOM character sequence used to select the set. Hex string expression.

<i>label</i> \$	Name of variable that will contain returned label value.
<i>point</i>	Data point within the <i>dataset</i> .
<i>strvar</i> \$	String containing the chart data.

Description Use the **CHART** directive to create two and three dimensional chart illustrations in a graphical application. The chart type and visual look information is determined by the **FMT=** option (several chart types are available). Chart values can have impact on the resolution ability of the chart; e.g., it cannot reliably produce pie slices that are thinner than 0.1 percent of the total pie chart value range.



Note: This feature requires ProvideX *Charting Control* activation. Refer to the ProvideX website for licensing information.

Chart Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a chart object are described in [Chapter 7. Control Object Properties, p.704](#).

Format 1: Define/Create

```
CHART ctl_id,@(col,ln,wth,ht),[,ctrlopt]
```

Use this format to create the chart. The value in *ctl_id* is a unique identifier that is used to generate a CTL value whenever the chart is selected and changed.

Format 2: Remove

```
CHART REMOVE ctl_id[,ERR=stmtref]
```

Use this format to delete an entire chart.

Format 3: Disable/Enable

```
CHART {DISABLE | ENABLE} ctl_id[,ERR=stmtref]
```

If the mouse is clicked on a chart while it is *enabled*, then the program receives a CTL event based on the *ctl_id* of the chart. This does not occur when the chart is *disabled*.

Format 4: Hide/Show

```
CHART {HIDE | SHOW} ctl_id[,ERR=stmtref]
```

With the **CHART HIDE** format, the chart remains active, but is not displayed. It is still accessible to program(s). Use the **CHART SHOW** to restore the user's display and access.

Format 5: *Loading Data into a Chart*

CHART LOAD *ctl_id, strvar\$* [,ERR=*stmtref*]

Use this format to load chart data into a chart. The last character in *strvar\$* identifies the separator for the data set while the value defined by **SEP=** when the chart is created, is used to identify the different data points within each set. Legend labels can be specified as the first element in a dataset definition followed by an equal sign (=); e.g.,

```
CHART LOAD 10, "100,200,300,400/150,250,350,450/"
```

This loads a chart with two data sets consisting of four points (or data elements).

```
CHART LOAD 10, "Last Year=100,200,300,400/This Year=150,250,350,450/"
```

This includes legend labels for the first example.

Format 6: *Clearing Data from a Chart*

CHART CLEAR *ctl_id* [,ERR=*stmtref*]

This clears the data values from the chart but not the titles and labels.

Format 7: *Clearing Data & Titles from a Chart*

CHART DELETE *ctl_id* [,ERR=*stmtref*]

This clears all of the data values and all of the titles from the chart.

Format 8: *Retrieving Values from a Chart*

CHART FIND is used to retrieve values from a chart based on the specified dataset and data point. The format is described below.

CHART FIND *ctl_id, dataset, point, numvar* [,ERR=*stmtref*]

The data is read from the chart and returned in *numvar*.

CHART FIND *ctl_id, dataset, point, label\$* [,ERR=*stmtref*]

This format retrieves the label in *label\$*. Legend labels are determined for a specified *dataset* when the *point* value is set to zero. Point labels are determined for a specified point when the *dataset* value is set to zero. The following example reads the value and label for the second point of the first dataset:

```
CHART FIND 10,1,2,RETURN_VALUE
```

```
CHART FIND 10,1,0,LEGEND_LABEL_1$
```

```
CHART FIND 10,0,1,POINT_LABEL_1$
```

Format 9: *Reading Selected Set from a Chart*

CHART READ *ctl_id, dataset, eom\$* [,ERR=*stmtref*]

Use **CHART READ** upon receiving a CTL notification that the control was selected with the mouse. The *eom\$* character will yield \$01\$ indicating that the dataset was selected by a mouse click.

Format 10: *Updating Existing Values in a Chart*

CHART WRITE is used to change existing values in a chart based on the specified dataset and data point.

CHART WRITE *ctl_id,set,point,data\$[,ERR=stmtref]*

This format updates the data for an individual point.

CHART WRITE *ctl_id,set,point,label\$[,ERR=stmtref]*

This format changes chart labels. Legend labels are updated for a specified dataset when the point value is set to zero. Point labels are updated for a specified point when the dataset value is zero.

CHECK_BOX Directive

Control Check Box

Formats

1. *Define/Create*: `CHECK_BOX [*]ctl_id,@(col,ln,wth,ht)=contents$[,ctrlopt]`
2. *Remove*: `CHECK_BOX REMOVE [*]ctl_id[,ERR=stmtref]`
3. *Disable/Enable*: `CHECK_BOX {DISABLE | ENABLE}[*]ctl_id[,ERR=stmtref]`
4. *Hide/Show*: `CHECK_BOX {HIDE | SHOW} [*]ctl_id[,ERR=stmtref]`
5. *Force Focus*: `CHECK_BOX GOTO [*]ctl_id[,ERR=stmtref]`
6. *Signal on Focus*: `CHECK_BOX SET_FOCUS ctl_id,ctl_val[,ERR=stmtref]`
7. *Logical On/Off*: `CHECK_BOX {ON | OFF} [*]ctl_id[,ERR=stmtref]`
8. *Read Activation Status*: `CHECK_BOX READ [*]ctl_id,state$[,mode$][,ERR=stmtref]`
9. *Update*: `CHECK_BOX WRITE [*]ctl_id,state$[,ERR=stmtref]`

Where

- * Optional. Use a leading asterisk to denote a *global* check box.
- @(col,ln,wth,ht) Position and size of the check box region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines. Note that width and height are for the total area (box plus text/description). Use line value -1 to display the check box on the tool bar.
- contents\$ Text/pictures to appear on the check box. {bitmap} and {icon} images are supported. String expression. See [CHECK_BOX contents\\$, p.49](#)
- ctl_id Unique logical identifier for the check box (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the **F4** key) or [Negative CTL Definitions, p.817](#). Use this value with the apostrophe operator to access various [Check Box Properties](#).
- ctrlopt Control options. Supported options for **CHECK_BOX** include:
 - ERR=stmtref** Error transfer
 - FNT="font,size[,attr]"** Font name, size, optional properties
Refer to the ['FONT' Mnemonic, p.609](#) for details.
 - MSG=text\$** Message string.
 - MNU=ctl** CTL value associated with right-click menu event.
 - OPT=char\$** (See [CHECK_BOX OPT= Settings, p.48.](#))
 - OWN=name\$** Name assigned for automated testing of this control.
 - TBL=char\$** Single character translation
 - TIP=text\$** Mouse pointer message.
To change the colour, refer to the ['TC'= System Parameter, p.689](#).
- ctl_val CTL value to generate when focus goes to the input field.
- mode\$ String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to toggle the check box (\$01\$ for **MOUSE-CLICK** or \$0D\$ for **Enter**).

- state* Current state of the check box (0=OFF, 1=ON).
- stmtref* Program line number or label to transfer control to.

CHECK_BOX OPT= *Settings*:

Available attribute/behaviour settings are listed below. Some characters may be combined. Invalid settings are ignored.

- "<" *Bitmap Left*. Places bitmap left of text.
- ">" *Bitmap Right*. Places bitmap right of text.
- "^" *Drop-down*. Adds drop-down functionality.
- "*" *Default*. Defines check box as default.
- "B" *Bitmap*. Has a bitmap whose width is divided into four images. Use this attribute to custom design check boxes of any colour, style or shape by controlling the bitmap image that appears. Each of the four divisions represents what a button will look like in a particular state:
 - 1st quarter: Bitmap image when button is disabled.
 - 2nd quarter: Bitmap image when button is in normal (released) state.
 - 3rd quarter: Bitmap image when the mouse is over the button.
 - 4th quarter: Bitmap image when the button is pressed.
- "d" *Permanently Disabled*. Checkbox is grayed out and cannot be enabled.
- "D" *Initially Disabled*. Checkbox is initially grayed out.
- "F" *Flat*. Check box shows no raised outline unless the mouse is over the button or the button is pushed.
- "f" *Flat-No Shift*. Same as "F", but will not shift when pressed.
- "G" *Global*. Keep active when focus changes to new/non-concurrent window. When using secondary commands (**REMOVE** or **SET_FOCUS**) on controls created with **OPT="G"** identify the control by prefixing the CTL value with an asterisk.; e.g.,
`CHECK_BOX 100 ,@(10,10,10,1)="Global" ,OPT="G"`
`CHECK_BOX REMOVE *100`
- "h" *Permanently Hidden*. Checkbox cannot be shown.
- "H" *Initially Hidden*. Checkbox is initially hidden.
- "P" *Sticky - Pressed*. Button remains in the "pressed" position until next selection.
- "S" *Signal Only*. ProvideX generates a CTL value, but does not shift focus to the check box automatically (the default), but only when focus is explicitly passed to it. Use this to have a check box act like a function key.
- "s" *Scroll*. Check box can scroll within a resizable/scrollable dialog box.
- "T" *Transparent*. Check box is "see-through" to window data behind.
- "U" *Underscore*. Text is underlined.
- "V" *Hovertext*. Indicates that text will change color when mouse is over the check box.
- "Y" *System Tray*. Places an icon in the *Taskbar Notification Area*.

Combined options can be used to create several different check box types. The "f", "T", and "U" options provide the ability to turn check boxes into *hotspots*, which allows for clickable areas on bitmaps or hyperlinked text in dialogues; e.g.,

- "VTf" Creates a general hotspot.
- "\VUTf" Creates an HTML-like hotspot (e.g., URL hyperlink).
- "F^" Creates a word-style toolbar with drop list

Description

Use the **CHECK_BOX** directive to create/control a check box object on the screen or to place an icon in the **Taskbar Notification Icon** (see *User's Guide*, p.155). The user can toggle check boxes between two states: **ON** to *check* the option or **OFF** to *uncheck* it. (If you need a *third* state for a check box, refer to the **TRISTATE_BOX Directive**, p.344.)

CHECK_BOX contents\$

The *contents\$* string expression defines the text or picture to appear on the check box. In the text, you can use an ampersand "&" preceding a character to identify it as a hot key the user can press in conjunction with the **Alt** key to activate the check box from the keyboard. By default, the text is displayed to the right of a check box. Add a colon (:) to the end of the text field to force the text to display to the left side.

Using Images

When adding an image to a check box, enclose the image name in curly braces. Use a leading exclamation point (!) to identify the image as internal, or specify the relative path and filename to access an image file that is external. There are no icons in the ProvideX executable and ProvideX does not support retrieving icons from either resource libraries or other system DLLs /executables. For more information on the options available for displaying internal/external images and the recognized image file types, see **Images and Icons**, p.153 in the *User's Guide*.

When you use text as well as images, the relative positions of the image and the text set their relative placement. The following are example *contents\$* expressions:

```
"{!Add}Add" ! Displays the {!Add} bitmap in front of the text "Add"
"Delete{!Del}" ! Displays the {!Del} bitmap after the text "Delete"
```

If you enclose two images separated by a pipe | vertical bar in a single set of curly braces, the first will be displayed when the **CHECK_BOX** state is 0 (zero for OFF/normal state), the second when the **CHECK_BOX** state is 1 (ON); e.g.,

```
"{!Stop|C:\MYBMP\Go}"
```

You can also use the OPT="B" clause for a *Bitmap Button* to display different images for different states.

Check Box Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a check box object are described in [Chapter 7. Control Object Properties, p.704](#).

Format 1: *Define/Create*

CHECK_BOX [*]*ctl_id*,@(col,ln,wth,ht)=contents\$[,*ctrlopt*]

Use this format to create a check box. The *ctl_id* is a unique value that is used to generate a CTL value whenever the user toggles the box; e.g.,

```
0010 REM Create check box, CTL=100, Text="Post Month End", ALT-P to select box
0020 CHECK_BOX 100,@(60,20,20,2)("&Post Month End"
```

If the *ctl_id* has a leading *asterisk* *, the check box is considered global (not tied to a specific window).

Format 2: *Remove*

CHECK_BOX REMOVE [*]*ctl_id*[,ERR=*stmtref*]

Use the **CHECK_BOX REMOVE** format to delete a check box. Note that, by default, all check boxes which are not global are deleted when a window is removed or dropped, and on a **BEGIN**; e.g.,

```
0030 CHECK_BOX REMOVE *17000 ! Removes global check-box 17000
```

Global check boxes can be removed using the above syntax, or cleared using the **START** directive.

Format 3: *Disable/Enable*

CHECK_BOX {DISABLE | ENABLE} [*]*ctl_id*[,ERR=*stmtref*]

Use the **CHECK_BOX DISABLE** format to gray out a check box so that it will be visible but *inaccessible* to users. To reactivate it, use **CHECK_BOX ENABLE**; e.g.,

```
0030 CHECK_BOX ENABLE 100
```

Format 4: *Hide/Show*

CHECK_BOX {HIDE | SHOW} [*]*ctl_id*[,ERR=*stmtref*]

Use the **CHECK_BOX HIDE** format to keep a check box from being displayed. Use the **CHECK_BOX SHOW** format to restore the display of a hidden check box.

Format 5: *Force Focus*

CHECK_BOX GOTO [*]*ctl_id*[,ERR=*stmtref*]

Use the **CHECK_BOX GOTO** format to reactivate and force focus to a check box, ready for the next user action; e.g.,

```
0030 CHECK_BOX GOTO 110
```

Format 6: *Signal on Focus*

CHECK_BOX SET_FOCUS *ctl_id*,*ctl_val*[,ERR=*stmtref*]

Use the **CHECK_BOX SET_FOCUS** format to define an alternate CTL value to generate whenever focus shifts to the check box.

Format 7: Logical On/Off

CHECK_BOX {ON | OFF} [*]ctl_id[,ERR=stmtref]

Use the **CHECK_BOX {ON | OFF}** format to make it appear that a check box is depressed or released; e.g.,

```
0030 CHECK_BOX OFF 110,ERR=0040
```



Note: This does not alter its state; i.e., "checked" or "unchecked". Use **CHECK_BOX WRITE** to do that.

Format 8: Read Activation Status

CHECK_BOX READ [*]ctl_id,state\$[,mode\$][,ERR=stmtref]

Use the **CHECK_BOX READ** format to receive the current *state* of the check box ("0"= off and "1"= on). Use the second (optional) string variable to have ProvideX return the user's method of toggling the check box. Once read, this field is reset to \$00\$. Some of the possible values are:

\$01\$ for **MOUSE-CLICK**.

\$02\$ for **DOUBLE MOUSE-CLICK**.

\$0D\$ for **Enter**.

\$20\$ for **SPACEBAR** (and keyboard HotKey, as in the example below).

\$00\$ when the user exits the check box.

Read the user's selection(s) to make them available to your applications. The **ON_OFF\$** variable in the example below receives the current on/off state. For **STROKE\$** in the example below, ProvideX returns \$20\$ when the user makes the selection using either **SPACEBAR** or the &T hot key **Alt-T**.

```
0010 ! CHECK_BOX Example
0020 PRINT 'CS'; LIST
0030 CHECK_BOX 90,@(40,17,25,3)="&Toggle for Status"
0040 LET C_BX=90; LET C_BX'BACKCOLOUR$="LIGHT CYAN"
0050 SETCTL C_BX:READ_BOX
0060 CHECK_BOX GOTO C_BX
0070 PRINT @(40,24),"HotKey=<Alt-T>. Try Mouse and keyboard too. END=<F4>"
0080 GETINPUT:
0080:OBTAIN (0,SIZ=1,ERR=GETINPUT)@(0,0),'CURSOR'("off'),'ME',IN_VAR$,'MN';
0080:LET CT=CTL
0090 IF CT=4 THEN GOTO END
0100 READ_BOX:
0110 CHECK_BOX READ C_BX,ON_OFF$,STROKE$
0120 ! WRITE_BOX
0130 CHECK_BOX WRITE C_BX,ON_OFF$
0140 PRINT @(40,20),"Current Status: ",ON_OFF$
0150 PRINT @(40,21),"Key Stroke : ",HTA(STROKE$)
```

Format 9: Update

CHECK_BOX WRITE [*]ctl_id,state\$[,ERR=stmtref]

Use the **CHECK_BOX WRITE** format to update the check box's state; e.g.,

```
0100 ! WRITE C_BX
0110 LET ON_OFF$="0"
0130 CHECK_BOX WRITE C_BX,ON_OFF$
```

See Also

RADIO_BUTTON Control Radio Button, p.265

BUTTON Control Button, p.34

TRISTATE_BOX Control Tristate Box, p.344

Chapter 7. Control Object Properties, p.701.

CLEAR Directive

Reset Variables

Formats

1. *Clear, Reset*: **CLEAR** [EXCEPT] [*varlist*]
2. *Clear Composite String*: **CLEAR** *template*\$

Where:

template\$ Name of a variable **DIM**ensioned as a string template. String expression.

varlist List of variables. Optional.

Description

The **CLEAR** directive performs several functions:

1. Resets **PRECISION** to the default value of 2.
2. Clears the following local variables (global variables are not affected unless specified in *varlist*):
 - All if no *varlist* appears on directive
 - Only those in *varlist* if no **EXCEPT** clause given
 - All but those in *varlist* if **EXCEPT** clause present
3. Sets **ERR**, **RET**, and **CTL** to zero
4. Clears **FOR/NEXT**, **GOSUB/RETURN** stack
5. Resets the pointer to the first **DATA** item in the program.

Note: When executed within an object, the **CLEAR** directive will clear all of the object's properties, along with standard variables. See **Data Integration**, p.275 in the *User's Guide*.



See Also

BEGIN Reset Files and Variables, p.32,
RESET Reset Program State, p.288,
START Restart ProvideX, p.328,
ERR System Variable, p.560,
CTL System Variable, p.557,
RET System Variable, p.571.

Format 1: *Clear, Reset*

CLEAR [EXCEPT] [*varlist*]

The examples here show, respectively, clearing all variables, clearing only a selected list of variables, and **CLEAR**ing all **EXCEPT** a selected list of variables:

```
0100 CLEAR
1100 CLEAR A3$,B3$,C3,D3
2100 CLEAR EXCEPT CST_ID$,TX_VAL,TX_TBL${ALL}
```

Format 2: *Clear Composite String*

Use this format to clear the attributes of a variable **DIM**ensioned as a string template; e.g., **CLEAR** CUST\$ or **CLEAR** CUST.NAME\$. For more on string templates, see **DIM Define Arrays and Strings**, p.86.

CLIP_BOARD Directive

Use Windows Clipboard

Formats

1. Read Clipboard Data: **CLIP_BOARD READ** *var\$*[,ERR=*stmtref*]
2. Write Clipboard Data: **CLIP_BOARD WRITE** *text\$*[,ERR=*stmtref*]

Where:

- stmtref* Program line number or label to transfer control to.
- text\$* The data to place into the Windows clipboard.
- var\$* String variable. Receives the contents of the clip board.

Description



Note: This directive is for Windows or WindX only.

Use the **CLIP_BOARD** directive to have ProvideX application read or write to the Windows clipboard. Only *text* can be passed to/from the clipboard. You can read a maximum of 32000 bytes using ProvideX to access the clipboard. The write limit depends on the system.

ProvideX does not interpret the data in the clipboard. Because of this, carriage return \$0D\$ and line feed \$0A\$ characters are present in multi-line clipboard data when it is read. You should place these characters between the lines when writing to the clipboard.

Examples

```
1000 INPUT "Enter customer ID:",C$
1010 READ (1,KEY=C$)C.NM$,C.ADR1$,C.ADR2$
1020 CLIP_BOARD WRITE C$+" "+C.NM$+$0D0A$+C.ADR1$+$0D0A$+C.ADR2$
1030 PRINT "Your information is in the clipboard.."
```

CLOSE Directive

Close File

Format `CLOSE {(*) | (chan)}[, (chan)...[, ERR=stmtref]]`

Where:

- * Asterisk denotes "all **OPEN** local channels". `CLOSE (*)` resets all files then closes everything but channel 0 (unless the **'WK'** parameter is set). This closes all open windows and purges graphical controls from the system.

chan Channel or logical file number.

stmtref Program line number or label to transfer control to.

Description Use the **CLOSE** directive to close a given logical file number. The memory used for buffers and control information is returned to the system. Once closed, the file (channel) number can be reused for a different file.



Note: A **CLOSE** directive will delete a Memory file. See ***MEMORY* Create & Use Memory File, p.741**. When you **CLOSE (chan)**, ProvideX returns all memory that was occupied by the Memory file to the system.

The **ERR=** branch will be taken should an error occur during the close. If the application tries to close an unopened file, an error will only be generated if an **ERR=** branch is provided. If an **ERR=** branch is not provided, the close is assumed to have occurred previously and no error is generated.



Note: When executed within an object, **CLOSE (*)** will close any standard local files and files owned by the object. See **Data Integration, p.275** in the *User's Guide*.

See Also [OPEN Open for Processing, p.232](#)
[BEGIN Reset Files and Variables, p.32](#)
[START Restart ProvideX, p.328](#)
['WK' System Parameter, p.695](#)

Examples

```
CLOSE (30)
OPEN (30)"PRINT_DEVICE"
0500 LET CSTFILE=HFN; OPEN (CSTFILE)"CSTFILE"
0510 LET PRDFILE=UNT; OPEN (PRDFILE)"PRDFILE"
0520 LET ORDFILE=7; OPEN (ORDFILE)"ORDFILE"
0530 PRINT CSTFILE,PRDFILE,ORDFILE
0540 STOP
-:run
63 5 7
```

Either of the two statements below will close channels 63, 5 and 7. The second statement closes all other open local channels as well.

```
CLOSE (CSTFILE),(PRDFILE),(ORDFILE) ! Just closes channels 63,5,7
CLOSE (*) ! Closes ALL open file channels (i.e., 63,5,7 and 30)
```


CONTINUE Directive *Initiates Next Iteration of Loop*

Format **CONTINUE**

Description Use the **CONTINUE** directive to go immediately to the next iteration of a loop (or terminate the loop if it is the last iteration). The *CONTINUE label emulates a **CONTINUE** directive for use as a statement reference.

See Also [Labels/Logical Statement References , p.816](#)
Flow Overrides, *User's Guide*

Examples

```
0010 FOR I=-1 TO 3 STEP 1
0020 IF I<=0 THEN PRINT " TESTING ",; CONTINUE
0030 PRINT I,
0040 NEXT
-:run
TESTING TESTING  1 2 3-:

0110 FOR I=1 TO 100
0120 IF X$[I]<K$ CONTINUE
0130 PRINT X$[I]
0140 NEXT
```

CREATE TABLE Directive

Create Keyed File (EFF)

Format

```
CREATE TABLE filename$[,extkey_len][,key_def$][,max_recs][,rec_size][,fileopt]
```

Where:

filename\$ Name of the Keyed file to create. String expression.

extkey_len Numeric expression. Length of the external key for all records in the file (maximum 127 characters).

key_def\$ String expression defining the key. The Keyed file can be single- or multi-keyed. A key definition is made up of one or more key field definitions ranging 0 to 15 for FLR/VLR files or 0 to 255 for EFF files. Use integers for specific field numbers, or 0 *zero* for record-based offsets. The key definition formats are as follows:

Single key field:

```
[["keyname":]field:offset:len[:"attr" ]
```

Composite key fields (using the + operator):

```
[["keyname":]field:offset:len[:"attr" ]]+[field:offset:len[:"attr" ]]
```

Multi-keyed alternate key fields are comma-delimited:

```
[["keyname":]field:offset:len[:"attr" ]],[["keyname":]field:offset:len[:"attr" ]]
```

Where:

keyname Name of key assigned for use in **KNO=name**\$ options.

field Integer, 1 to number of fields (0 = record-based offset).

offset Starting position within the field (integer, 1 to 3839).

len Number of characters in the key field (integer, max. 127)

"attr" Attribute characters, see [Key Definition Attributes, p. 167](#)

: Colon - the separator for elements in a key segment.

The maximum total size for an external key is 127 characters. The maximum length for internal or alternate keys is 240 characters.



Note: The *outer* set of square brackets in the above formats are part of the syntax; the *inner* brackets indicate optional syntax items (i.e., the brackets enclosing the optional [:"attr"] are not part of the syntax).

max_recs Maximum number of records the file is allowed. Optional numeric expression. The default is zero (no limit). (Use a comma with no value to set the default.) If a positive value is supplied, ProvideX creates and pre-allocates disk space for the file. With a negative value, ProvideX allocates sufficient disk space for the file, but will set the *max_recs* count back to zero (unlimited).

rec_size Maximum size of the data portion of the record (excluding the key). A negative value creates a variable-length record (VLR) data file with the maximum record length equal to the positive value of this field. A positive value creates a fixed-length record (FLR) formatted file.

If you do not specify size, the default is VLR with a maximum record size of 256. The maximum block size for a VLR file is 31KB and the maximum record size is 31000 bytes. Attempting to create a VLR file with a record size more than 31000 bytes results in an FLR file with the requested record size.

fileopt Supported file options (see also, [File Options, p.810](#)):

BSZ=num Block size. Numeric expression (1 - 63).

ERR=stmtref Error transfer.

SEP=char\$ Default field separator character. Hex or ASCII string value.

OPT=char\$ Single character settings:

"C" - *Compressed*. Adds simple compression to record data.

"X" - *Extended Record Size*. Extends record sizes up to 2GB per record.

"0" - *Create VLR/FLR files (default if 'KF'=0)*

"1" - *Create EFF Files with 2GB limit*.

"2" - *Create EFF Files without 2GB limit (supported platforms)*.

"Z" - *Set ZLib Compression for VLR and EFF Files*.

Note: OPT="2" generates Error #99: Feature not supported on platforms that do not offer Large File Support (LFS). Using options "Z" and "C" together will result in an Error #32.



Description

Use the **CREATE TABLE** directive to create a file with one or more keys. If the first field in the directive after the filename is a number, ProvideX creates an *external* file key (i.e., an index to the file). If the first field in the directive is a key definition enclosed in square brackets [], then ProvideX uses only *internal* key fields instead.

Note: By default, the **CREATE TABLE** directive will create Enhanced File Format (EFF) files on platforms that support Large File System (LFS), 64-bit addressing; however, by setting OPT="0" in the syntax, **CREATE TABLE** can also be used to create variable-length record (VLR) data files or fixed-length record (FLR) formatted files. For more information on VLR/FLR, refer to the [KEYED Directive, p.166](#).



ProvideX considers the first key specified for an *EFF* file to be the primary key. Every record must have a unique primary key. You can have duplicate secondary keys from record to record. There is a maximum of 255 keys allowed on a file with a maximum of 255 data components making up these 255 keys. For *VLR/FLR* files, there is a maximum of 16 key fields allowed on a file with a maximum of 96 data components making up the 16 keys. There is no limit (other than the maximum of 96 key components) to the number of fields that comprise a key. *The initial implementation of EFF (in Version 6) is limited to 96 keys and 96 segments.*

If a given filename already exists, ProvideX returns Error #12: File does not exist (or already exists).

Keys have the following limits:

- Maximum external key size is 127 bytes.
- Maximum internal or alternate key size is 240 bytes.
- Maximum segment size is 127 bytes.

Enhanced File Format (EFF) Notes

EFF records are always variable length. The following limitations exist for EFF files:

1. 63Kb (64512 bytes) is the maximum block size.
2. The maximum record size is 64000 bytes. When using extended records the record size is limited to 32000 bytes when created, although records can be as large as 2GB.
3. EFF files do not support the multi-segmented techniques available for VLR files.
4. *Refer to chart below for sample file size limitations.* The block size determines the maximum file size. As of Version 6, EFF files will only use 3-byte pointers. The ability to use 4-byte pointers is part of the design to allow larger file sizes and will be enabled in a future version of ProvideX.

Block Size (KB)	3-Byte Pointers				4-Byte Pointers (<i>future</i>)			
	Max Pages per Invoice Segment	Max Addressible Pages	Size of File in MB	Size of File in GB	Max Pages per Invoice Segment	Max Addressible Pages	Size of File in MB	Size of File in GB
4	1,022	2,056,264	8,032	8	818	1,645,816	6,429	6
8	2,046	8,388,607	65,536	64	1,637	9,998,796	78,116	76
12	3,070	8,388,607	98,304	96	2,456	25,061,024	293,684	287
16	4,094	8,388,607	131,072	128	3,275	46,832,500	731,758	715
20	5,118	8,388,607	163,840	160	4,094	75,313,224	1,470,961	1,436
24	6,142	8,388,607	196,608	192	4,914	110,525,688	2,590,446	2,530
28	7,166	8,388,607	229,376	224	5,733	152,429,004	4,167,981	4,070
32	8,190	8,388,607	262,144	256	6,552	201,041,568	6,282,549	6,135
36	9,214	8,388,607	294,912	288	7,371	256,363,380	9,012,775	8,802
40	10,238	8,388,607	327,680	320	8,190	318,394,440	12,437,283	12,146
44	11,262	8,388,607	360,448	352	9,010	387,177,720	16,636,543	16,247
48	12,286	8,388,607	393,216	384	9,829	462,631,372	21,685,846	21,178
52	13,310	8,388,607	425,984	416	10,648	544,794,272	27,665,334	27,017
56	14,334	8,388,607	458,752	448	11,467	633,666,420	34,653,632	33,841
60	15,358	8,388,607	491,520	480	12,286	729,247,816	42,729,364	41,728
63	16,126	8,388,607	516,096	504	12,901	805,383,628	49,549,969	48,389

See Also

[KEYED Create Single/Multi-Keyed File, p.166](#)

[DIRECT Create File with Keyed Access, p.89](#)

[SYSTEM_JRNL File System Journalization, p.334](#)

[ADD INDEX Add Key to Keyed File, p.29](#)

[DROP INDEX Drop Key from Keyed File, p.103](#)

[RENAME..INDEX Rename Keys in Keyed File, p.285](#)

[SORT Create File for Sorting, p.327](#)

[Accessing Data Files, p.22](#)



CUSTOM_VBX Directive

Create/Control VBX

Formats

1. Define VBX, Logical ID: **CUSTOM_VBX** *id,@(col,ln,wth,ht),name\$,control\$[,fileopt]*
2. Define Event-CTL: **CUSTOM_VBX DEFCTL** *id,ctl_val,event\$[,ERR=stmtref]*
3. List Event Names: **CUSTOM_VBX DEFCTL** *id,*,list\$[,ERR=stmtref]*
4. Add Item to List: **CUSTOM_VBX LOAD** *id,index,val\$[,ERR=stmtref]*
5. Remove List Item: **CUSTOM_VBX LOAD** *id,index,*[,ERR=stmtref]*
6. Read Property: **CUSTOM_VBX READ** *id,property\$,var\$[,array_element][,ERR=stmtref]*
7. List Properties: **CUSTOM_VBX READ** *id,*,list\$[,ERR=stmtref]*
8. Write Property: **CUSTOM_VBX WRITE** *id,property\$,val\$[,array_element][,ERR=stmtref]*
9. Disable/Enable: **CUSTOM_VBX {DISABLE | ENABLE}** *id[,ERR=stmtref]*
10. Force Focus: **CUSTOM_VBX GOTO** *id[,ERR=stmtref]*
11. Remove VBX: **CUSTOM_VBX REMOVE** *id[,ERR=stmtref]*
12. Hide/Show VBX: **CUSTOM_VBX {HIDE | SHOW}** *id[,ERR=stmtref]*

Where:

@(col,ln,wth,ht)	Position and size of the CUSTOM_VBX region.
control\$	Name of the control to use in the file. String expression.
ctl_val	CTL to be generated when an event occurs.
event\$	Name / number of the VBX event.
fileopt	Supported file options (see also, File Options, p.810): ERR=stmtref Error transfer KEY=char\$ Hot key MSG=text\$ Message line OPT=char\$ Settings: "s" - Scroll, "D" - Disabled, "H" - Hide. TBL=char\$ Single character translation
id	Unique logical identifier for the CUSTOM_VBX control object.
index	Array index. Numeric expression, integer.
list\$	String variable for receiving list of event / property names.
name\$	Name of the VBX file containing the logic. String expression.
property\$	Name / number of the VBX property.
val\$	Value of an item / property.
var\$	Variable to receive current value of the property.



Deprecated. VBX support is discontinued as of ProvideX *Version 7*. Execution of this directive will result in an `Error #99 Feature not supported`. Refer to the *Version 6 Language Reference* for complete documentation.

CWDIR Directive

Change Working Directory

Format `CWDIR new_dir$[,ERR=stmtref]`

Where:

new_dir\$ String expression. Name of the new directory in which you want to work or run applications. Include a disk letter and colon ":" to change to another drive.

stmtref Program line number or label to transfer control to.

Description Use the **CWDIR** directive to change from the current working directory to a new one. If the specified directory name does not exist, ProvideX returns an Error #12: File does not exist (or already exists).

See Also [LWD System Variable, p.565](#),
[HWD System Variable, p.563](#)

Examples

```
0010 CWDIR "C:\Program Files\"+C$+"Programs"  
0020 RUN "SOMETHING"  
0030 CWDIR HWD ! Return home
```

DATA Directive

Define Data Elements

Format **DATA** *expression*[\$][, ...]

Where:

expression[\$], ... A series of string or numeric expressions.

Description Use the **DATA** directive to define the values for the **READ DATA** directive. The values to be read can include constants, variables, and/or expressions. You can use a formatted IOList with a **READ DATA** statement. When ProvideX executes the **READ DATA** directive, it evaluates each expression in order and places its value into the corresponding variable(s) defined in the **READ DATA** directive.

When ProvideX progresses through reading the data, it increments an internal pointer to the next data expression. When the end of **DATA** statement is reached, ProvideX proceeds to the next **DATA** statement in the program. ProvideX returns

Error #2: END-OF-FILE on read or File full on write
when no further **DATA** statements exist in the program.



Note: You cannot use the **DATA** directive in a compound statement.

See Also

[READ DATA Read Data from Program, p.273](#),
[RESTORE Reset Program Data Position, p.289](#),
[BEGIN Reset Files and Variables, p.32](#)
[LOAD Read Program into Memory, p.194](#)

Example

```
00010 DATA 1, "Dog"
00020 DATA 2, "Cat"
00030 DATA 3, "Pig"
00040 READ DATA n,x$,ERR=DONE
00050 PRINT n,x$
00060 GOTO 0040
00070 DONE: PRINT "done"
00080 END
```

```
RUN
1Dog
2Cat
3Pig
done
```

DAY_FORMAT Directive

Specify DAY Format

Formats

1. Set Date Format Mask: **DAY_FORMAT** [*new_fmt*]
2. Read Date Format Mask: **DAY_FORMAT READ** *current_fmt*

Where:

- new_fmt* String expression containing the format for the variable **DAY**.
Optional. Defaults to "MM/DD/YY".
- current_fmt* String expression (max 8kb) containing current **DAY_FORMAT** setting.

Description

Use the **DAY_FORMAT** directive to change the format mask for the **DAY** variable. The format string contains the new format for the **DAY** variable. Characters in the format string control the following:

- DD Insert current day
- MM Insert current month
- YY Insert current year (last two digits only)
- YYYY Insert four digit year
- AA Insert current year using 00-99 for 1900 through 1999, A0-A9 for 2000 through 2009, B0-B9 for 2010 through 2019, etc.

All other characters in the mask are returned as literals in the **DAY** variable. The **DAY_FORMAT** stays in effect until a **START** directive is executed.

The **DAY_FORMAT** directive mask can contain the letters AA (e.g., "MM/DD/AA"). The format AA indicates a packed two-character year field. ProvideX returns \$A0\$ for the year 2000, \$A1\$ for 2001, \$B0\$ for 2010, etc. up to \$Z9\$ for the year 2269. This format assists with conversion issues in legacy applications.

See Also

[DAY System Variable, p.557](#),
[DTE\(\) Function, p.422](#),
[JUL\(\) Function, p.463](#).

Examples

Example 1:

```
0010 PRINT DAY
0020 DAY_FORMAT "YYYY MM/DD"
0030 PRINT DAY
->RUN
11/15/00
2000 11/15
```

Example 2:

```
0010 LET X$="01/01/A0"
0020 DAY_FORMAT "MM/DD/AA"
0030 PRINT JUL(X$)
0040 PRINT DTE(10957:"%Y %M1 %D")
->RUN
10957
2000 January 1
```


DEF CLASS Directive

Define Object Class

Formats	DEF CLASS <i>class\$</i> [UNIQUE][CREATE <i>label</i> [REQUIRED]][DELETE <i>label</i> [REQUIRED]]
	<i>Where:</i>
	class\$ Class name that will refer to this type of object.
	CREATE <i>label</i> Optional keyword with line label for override of ON_CREATE logic
	DELETE <i>label</i> Optional keyword with line label for override of ON_DELETE logic.
	REQUIRED Optional keyword to force execution of ON_CREATE/ON_DELETE logic.
	UNIQUE Optional keyword to force a single instance of an object in memory.

Description **DEF CLASS** is a directive used in **Object Oriented Programming (OOP)** to declare the start of a **Class Definition**. The **DEF CLASS** statement comprises a modular construct that includes a series of OOP directives concluding with an **END DEF** directive to mark the end of the class definition.

class\$ specifies the class name that will refer to this type of object. Class names are case-insensitive and forward/backward slashes are considered equivalent. Duplicate names are not allowed within the system. An object declared as **UNIQUE** will have a single instance created, and any subsequent attempt to create an instance returns the same object identifier and increments the object reference count by one.

Optional initialization and/or cleanup routines can be placed under the default ON_CREATE and ON_DELETE labels. You can override these labels via **CREATE** *label* and **DELETE** *label* clauses in the class definition. Normally, object creation/deletion logic is invoked when an object of this *specific class* is created/destroyed. That means, if you have ON_CREATE logic for an object *A* and object *B* inherits it, then the ON_CREATE for object *A* will not be executed on creation of object *B*. You can force creation and/or deletion logic to be executed on inheritance by including the keyword **REQUIRED**.

Class Definition

In order to use an object, you must first define its characteristics. Each object is defined with a **DEF CLASS** statement. This basic construct is outlined below.

```
0010 DEF CLASS "class$" ...
0020 PROPERTY prop1, prop2, ...
0030 LOCAL prop1, prop2, ...
0040 FUNCTION method (param) "
0050 LIKE "otherclass"
0060 PROGRAM "interface_prog"
0070 PRECISION nnn
0080 END DEF
```

All properties are declared by the **PROPERTY** directive. Methods available for an object are defined via the **FUNCTION** directive. For more information on ProvideX OOP syntax, refer to the ProvideX *User's Guide*, [Chapter 10](#).

Example

The following is an example of the definition of the object "Customer":

```
0010 DEF CLASS "Customer"
0020 LIKE "Company"
0030 PROPERTY Limit,LastInvDate$
0040 FUNCTION Invoice();"Invoice"
0050 FUNCTION Edit();"Edit"
0060 END DEF
0110 On_Create: GOSUB Where_are_we; RETURN 1
0120 On_Delete: GOSUB Where_are_we; RETURN 1
0210 Invoice: GOSUB Where_are_we; RETURN 1
0220 Edit: GOSUB Where_are_we; RETURN 1
0310 Where_are_we:
0320 MSGBOX "In the Customer "+FNLabelName$+" logic","FYI"
0330 RETURN
8010 def fnLabelName$=mid(pgm(-3),pos("; "=pgm(-3)+";")+1)
```

See Also

[Object Oriented Programming, p.22](#)
[FUNCTION Declare Object Method, p.137](#)
[LIKE Inherit Properties, p.174](#)
[LOCAL Designation of Local Data, p.197](#)
[PROGRAM Create/Assign Program File, p.259](#)
[PRECISION Change Current Precision, p.248](#)
[PROPERTY Declare Object Properties, p.261](#)
[Data Integration, User's Guide](#)



DEF GID/UID Directives

Define Group/User ID

Formats

1. *Define Group ID*: **DEF GID=** *groupID* | *groupname*\$ [,**ERR=***stmtref*]

2. *Define User ID*: **DEF UID=** *userID* | *username*\$ [,**ERR=***stmtref*]

Where:

groupID Numeric UNIX/Linux group ID number

groupname\$ UNIX/Linux group name.

userID Numeric UNIX/Linux user ID number

username\$ UNIX/Linux user name.

stmtref Program line number or label to transfer control to.

Description

These directives allow you to override the effective UNIX/Linux user or group ID of a session, if OS security allows it. If the *value* is *, the system reverts back to the original effective user or group ID.



Note: Whether or not you can switch to a different user is governed by the operating system. If the OS does not allow the change, an error will be returned.

DEF FN Directive*Define Function*

Formats

1. *Single Line*: **DEF FN***name*[\$]([**LOCAL**]*argvar1* [,*argvar2*, ...])=*expression*[\$]
2. *Multi-Line*: **DEF FN***name*[\$]([**LOCAL**]*argvar1* [,*argvar2*, ...])
RETURN *expression*[\$]
END DEF

Where:

- argvar ...* Comma-separated list of variables that correspond to arguments passed to the function or returned in the expression.
- expression*[\$] String or numeric expression. In multi-line functions the *expression* is the value returned. In single-line functions, the expression defines the value of the function.
- FN***name*[\$] Name of the function with **FN** prefix. Use a valid string or numeric variable name, e.g., **FNX** or **FNABC**\$.
- LOCAL** Optional keyword. Indicates that an argument variable is local to the function. Use **LOCAL** to prevent permanent changes to program variables. ProvideX defers processing the **LOCAL** clause in functions until all arguments are parsed.

Description

Use the **DEF FN** directive to define single- or multi-line functions. These functions are considered string or numeric depending on the type of variable used in the function name.

In Execution mode, ProvideX skips past the function without executing it. ProvideX resumes after the **DEF FN** directive, executing the rest of the code until it reaches the statement that invokes **FN***name*()... whereupon it applies the defined function.

Format 1: *Single Line Function*

DEF FN*name*[\$]([**LOCAL**]*argvar1* [,*argvar2*, ...])=*expression*[\$]

In a single-line function assignment, *expression* defines the value to be returned by the function. Include the optional keyword **LOCAL** to define argument variables as local.

Examples

```
0010 DEF FNX(A,B)=A^B
0020 LET A=1, B=2
0030 PRINT A,B,FXN(3,4),A,B
0040 STOP
->RUN
1 2 81 3 4
```

When `FNX` is invoked (as in `PRINT FNX(3 , 4)`), the values in the *variable list* are assigned to the variables in the function definition.



Note: The variables in the function definition's argument list represent actual variables in the program. Their values are subject to change every time the function is invoked. Unless you define variables as **LOCAL**, changes to the variables in your program will be permanent. In the example above, variables `A` and `B` in the program are changed by `FNX(3,4)`, because they aren't defined as **LOCAL**.

Format 2: *Multi-Line Function*

```
DEF FNname[$]([LOCAL ]argvar1[,argvar2, ... ])
RETURN expression[$]
END DEF
```

When ProvideX encounters a multi-line definition, execution skips the subsequent statements until an **END DEF** directive is found. (Again, the function is not executed until it is invoked.) Include the optional keyword **LOCAL** to define variables as local.

The **RETURN** expression specifies the value to be returned by the multi-line function. To generate an error value in a multi-line function, use the **ESCAPE** directive followed by your given value.

Example

```
0010 DEF FNX(A,B)
0020 IF A<0 OR B<0 ESCAPE 40 ! Variables' values must be >=0
0030 RETURN SQR(A^2+B^2) ! Length of hypotenuse
0040 END DEF
0100 PRINT FNX(7,8)
->RUN
10.63
```

DEF MSG Directive

Define Temporary Message

Formats

1. *Override Message for Number:* **DEF MSG**(*err_msg*)="message"
2. *Define Message & Name:* **DEF MSG**(*name*\$)="message"
3. *Remove Message Override:* **DEF MSG**(*err_msg*) **DELETE**
4. *Remove Message Name:* **DEF MSG**(*name*\$) **DELETE**

Where:

message String containing message text associated with *name*\$ or to override the message text in *nnn*.

name\$ New system message name.

err_msg Number of the error message to return. Numeric expression. If *err_msg* is a positive integer, it returns the associated message, as described under [Error Codes and Messages, p.828](#). If *err_msg* is -1, it returns extended or external error information.

Description

The **DEF MSG** directive allows you to change system messages on-the-fly. It can be used to temporarily override values returned by the **MSG()** function for specific message numbers. It also allows the creation of text-based message names.

See Also

[MSG\(\) Function, p.484](#)

[MSGBOX Display PopUp Message Box, p.212](#)

[MESSAGE_LIB Establish Message Library, p.208](#)

[Error Codes and Messages, p.828](#)

Examples

```
DEF MSG(11)="Your own syntax message"  
DEF MSG(-1)DELETE ! Clear current MSG(-1)
```

DEF OBJECT Directive

Define Object

Formats

1. *Invisible Object*: **DEF OBJECT** *obj_id*, *obj_name*\$ [;**LICENSE**=*key*] [;**FINALIZE**=*method*] [,**ERR**=*stmtrf*]
2. *Visible Object*: **DEF OBJECT** *obj_id*,@(col,ln,wth,ht) {, |=} *obj_name*\$ [;**LICENSE**=*key*] [;**FINALIZE**=*method*] [,**ERR**=*stmtrf*]

Where:

@(col,ln,wth,ht)	Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.
<i>obj_id</i>	Numeric variable that will be used to save the object reference.
<i>key</i>	Optional license key. See Using Licensed Objects, p.72 .
<i>method</i>	Optional finalize method. See Finalize Method, p.73 .
<i>obj_name</i> \$	String expression identifying the object to be referenced, as well as any object specific parameters. See <i>Object Name Contents</i> , below.
<i>stmtrf</i>	Optional program line number or label to transfer control to.

Description

The **DEF OBJECT** directive is used primarily to create a new instance of a specified COM object by placing a reference to the object *obj_name*\$ into the supplied numeric variable *obj_id*. This is a feature of the ProvideX *Event Handling Interface* and *Component Object Model* (COM). (This directive is also used for implementation of the ProvideX ***SYSTEM** and ***XML** interfaces).

Object Name Contents

The **DEF OBJECT** *obj_name*\$ string may contain one of the following:

*	<i>Asterisk</i> displays all registered COM controls on system.
<i>CLSID</i>	Hexadecimal class identifier for object in the format {hhhhhhhh-hhhh-hhhh-hhhhhhhhhhhh}.
<i>progID</i>	Programmatic identifier name; e.g., Word.Document.
[DESIGN] <i>filename</i> .XML	Indicates that a previously saved OLE/ActiveX control definition should be loaded from the specified .XML file.
[DCOM] <i>server;name</i>	Indicates that the object is located on a remote system. <i>server</i> parameter is optional, and can be specified either by name, or by IP address. If not supplied then the object is considered local. <i>name</i> parameter is the <i>CLSID</i> or <i>progID</i> for the object.
[FILE] <i>x:\filename</i>	Indicates that the object should be created using the specified file name. An example of this would be a Microsoft Word document file.
[GLOBAL] <i>name</i>	Indicates that a reference to an object exposed by the use of <code>PvxMakeGlobal</code> should be obtained. The <i>name</i> parameter is the name used to expose the object.

[GETOBJECT] <i>name</i>	Indicates that ProvideX should bind to a running instance of the named object. <i>name</i> is the <i>CLSID</i> or <i>progID</i> for the object or a file-based moniker. File monikers are commonly used when dealing with WMI, LDAP, and related services in Windows.
[REGISTER] <i>x:\filename;name</i>	Ensures that the object information is properly registered before attempting to create an instance of the object. <i>x:\filename</i> parameter is the name of the executable file or library that exposes the automation object. <i>name</i> parameter is the <i>CLSID</i> or <i>progID</i> for the object.
[RUNNING] <i>name</i>	Indicates that ProvideX should bind to a running instance of the named object, where <i>name</i> is given as <i>CLSID</i> or <i>progID</i> . An error occurs if the object is not currently running.
[RUNNING OR NEW] <i>name</i>	Indicates the same functionality as [RUNNING] syntax; however, if the object is not currently running, ProvideX attempts to create a new instance of the named object.
[PICTURE] *	Indicates creation of an empty IPicture object.
[PICTURE] <i>filename</i>	Indicates that an IPicture object should be created and the specified image file should be loaded by the object.
[PICTURE] [#] <i>name</i> ;{ BMP CUR ICO }	Indicates that an IPicture object should be created and specified resource contents loaded by the object. For numeric resources, <i>name</i> should be prefixed with a #; e.g., #101. The image type is specified as the second parameter, and indicates the resource group that contains the desired resource.
*XML	Initializes ProvideX interface for parsing and serializing XML documents. See *XML , p.764.
*SYSTEM	Initializes object for simplifying event handling in ProvideX. See *SYSTEM , p.751.

Using Licensed Objects

Redistribution of a third party COM control may sometimes require the use of a *license file* (usually identified by a .lic extension) for permitting access to the control. The following steps outline how to extract the license key from the license file, and how to make it available in a run-time environment:

1. On the system where the COM object and license file have been installed, obtain a reference to the object without specifying the license information.
2. Query the `PvxLicense$` property of the object for the license key. If the object is licensed, the key data is returned as a string of hex characters.
3. Add the `LICENSE=key` data to the **DEF OBJECT** statement.

Finalize Method

A method of the object instance may be assigned to run upon release of the object. It should not require any parameters to be passed to it. This purpose of this is to simplify handling of an automation "server", such as a Word or Excel application, that requires a `Quit()` method in order to shut down. This *finalize* method removes the need for a `Quit()` (or similar method) to be called.

Format 1: Define Invisible COM Object

DEF OBJECT *obj_id,obj_name*[\$[,ERR=*stmtref*]

Use this format to create a link to an invisible object. This format associates the object with the current window, yet it is hidden from view; e.g.,

```
0010 DEF OBJECT this_com_id, "Mabry.SoundX"
```

Once the definition is complete, *this_com_id* will contain a handle to the object. This handle will be used to get and set *properties* and *methods* and service *events*. Note that since the value returned in *this_com_id* is a handle (memory pointer) to the object, it should not be changed by the application.

Format 2: Define Visible COM Object

DEF OBJECT *obj_id,@(col,ln,wth,ht) {, |=}**obj_name*[\$[,ERR=*stmtref*]

This format defines a visible object and associates it with a location in the current window. A comma or an equal sign may be used to assign coordinates; e.g.,

```
DEF OBJECT numvar,@(col,row,wide,high), "object_name", err=2000
DEF OBJECT numvar,@(col,row,wide,high)="object_name", err=2000
```

See Also

[DELETE OBJECT Remove Windows Object, p.84](#)
[ON EVENT Event Processing, p.228](#)
[Apostrophe Operator, p.823](#)
[COM Support, User's Guide](#)

Examples

Upon successful execution of the **DEF OBJECT** statement, ProvideX will place the object reference into the supplied numeric variable. Some examples of the **DEF OBJECT** statement include the following:

```
DEF OBJECT X, "*"
DEF OBJECT X, "Word.Application", ERR=*NEXT
DEF OBJECT X, @(1,1, 70, 20)="Word.Document"
DEF OBJECT X, "[dcom]MyServer;Shell.Explorer"
DEF OBJECT X, @(10, 2, 20, 10)="[file]c:\Documents and Settings\Default
    User\My Documents\test.doc"
DEF OBJECT X, "VCF1.VCF1Ctrl.1;License=8041207972768742028669631967"
DEF OBJECT X, "[running or new]Excel.Application"
```

The **DEF OBJECT** statement can also be used to bind child objects, which are returned as the result of either a property access or method call.

DEF systab= Directives

Define System Tables

- Formats
1. *Define Accent Conversion*: **DEF CVS**(*new_table*\$)
 2. *Define Date Table*: **DEF DTE**(*new_table*\$)
 3. *Define Lowercase Table*: **DEF LCS**(*new_table*\$)
 4. *Define Uppercase Table*: **DEF UCS**(*new_table*\$)

Where:

new_table\$ String expression. Contains the new definition (contents) of the table.

Description Use these **DEF** directives to define new system tables for *Accent Conversion*, *Date*, *Lowercase* and *Uppercase*.

See Also [CVS\(\) Function, p.412](#),
[DTE\(\) Function, p.422](#),
[LCS\(\) Function, p.472](#),
[UCS\(\) Function, p.546](#).

Format 1: Define Accent Translation Table

Use the **DEF CVS** directive to set the values for the accent conversion table. The value of each byte to be translated is used as an offset into the table. The character at the particular offset is used in place of the original character. For more information, refer to the "T" option under [Key Definition Attributes, p.167](#) and the [CVS\(\) Function, p.412](#).

Format 2: Define Date Table

The **DEF DTE** directive uses a string made up of 46 comma-delimited fields. These fields are used by the **DTE** function and should have the following values:

<i>Number of Fields</i>	<i>Field Contents</i>
12	Long form month names (%M1)
7	Long form day names (%D1)
2	Long form lower-case am/pm (%p1)
2	Long form upper-case AM/PM (%P1)
12	Short form month names (%Ms)
7	Short form day names (%Ds)
2	Short form lower-case am/pm (%ps)
2	Short form upper-case AM/PM (%Ps)

Formats 3 and 4: *Define Lowercase and Uppercase Tables*

The **DEF LCS** and **DEF UCS** directives both take a 256 character string and replace the standard case conversion table with the string's value. Whenever ProvideX converts the case of a character, it uses the character's binary value as an offset into the tables.



Warning: Be careful when changing the **UCS** conversion table because the ProvideX compiler uses this table to convert keywords to upper case before scanning its syntax tables. If you define an incorrect table, you may be unable to enter any subsequent commands in ProvideX. See the list of [Reserved Words](#) , p.827.

DEF sysvar= Directives

Define System Variables

- Formats
1. Set Contents of CTL Variable: **DEF CTL = num**
 2. Set Contents of ERR Variable: **DEF ERR = num**
 3. Set Contents of LFO Variable: **DEF LFO = num**
 4. Set Contents of LFA Variable: **DEF LFA = num**
 5. Set Contents of EOM Variable: **DEF EOM = strvar\$**
 6. Set Contents of RET Variable: **DEF RET = num**

Where:

- num** Numeric value for setting selected variable.
- strvar\$** String value for setting selected variable.

Description Use these **DEF** formats to define the contents of the specified system variable:

- DEF CTL** Numeric code (integer) that represents a signal of user input from the keyboard or mouse.
- DEF ERR** Numeric value (integer) that indicates the last system-detected error.
- DEF LFO** Channel/file number of the last file opened.
- DEF LFA** Channel/file number of the last file or device accessed.
- DEF EOM** End-Of-Message character string that ended last input.
- DEF RET** Operating system's error code associated with the last operating call.

See Also [CTL System Variable, p.557](#),
[ERR System Variable, p.560](#)
[LFO System Variable, p.564](#)
[LFA System Variable, p.563](#)
[EOM System Variable, p.559](#)
[RET System Variable, p.571](#).

DEFAULT Directive *Branch If No Matching Case*

Format **DEFAULT**; *logic*\$

Where:

logic\$ Procedure to handle default (undefined) cases in a case structure. It doesn't have to be on the same line as the **DEFAULT** directive, but it can be if you include the semicolon.

Description Use the **DEFAULT** directive to create branch points to handle situations where there is no corresponding **CASE**. If a matching case is not found and the **DEFAULT** is found, execution continues at this point.



Note: Refer to **SWITCH..CASE Branch Control, p.331**, for complete syntax.

See Also **SWITCH..CASE Branch Control, p.331**
BREAK Immediate Exit of Loop, p.33
CASE Define Branch Points, p.42.

Examples

```
00100 PROCESS_TAXCODE:
00110  LiquorTax=0,SalesTax=0,ServiceTax=0
00120  SWITCH UCS(TaxCode$)
00130  CASE "X","Z" ! two codes are tax exempt
00140  BREAK ! stop processing for case "X" here
00150  CASE "L" ! liquor pays all liquor,sales and service tax
00160  LiquorTax=cost*LiquorTaxRate
00170  ! no break here, logic falls through
00180  CASE "S" ! pays sales and service tax
00190  SalesTax=cost*SalesTaxRate
00200  ! no break here, logic falls through
00210  CASE "V" ! service tax
00220  ServiceTax=cost*ServiceTaxRate
00230  BREAK ! end processing for this case and any that fell through
00240  DEFAULT ! enter here if case not found
00250  MSGBOX "Unknown tax code","Error"
00260  END SWITCH
00270  TotalTax=LiquorTax+SalesTax+ServiceTax
00280  RETURN
```

DEFCTL Directive *Define/Redefine CTL Values*

Formats

1. *Define/Delete CTL Values*: **DEFCTL** [**WINDOW**] *eom\$*={*ctl_val* | *}
2. *Redefine CTL Value*: **DEFCTL** [**WINDOW** | **WINDOW+**] *ctl_val*={*alternate* | *}

Where:

- * Asterisk deletes previously defined **CTL** value (e.g., **DEFCTL** *eom\$*=*).
- alternate* Alternate integer to set as the **CTL** value. Numeric expression.
- ctl_val* **CTL** variable's value. Numeric expression, integer.
- eom\$* **EOM** (End-of-Message) character sequence. Hex string expression (e.g., \$0D\$ for the **Enter** key).
- WINDOW**[+] Optional keyword,
 - WINDOW**: **CTL** setting is for the current window only.
 - WINDOW+**: **DEFCTL** cascades to apply to lower level windows.

Description

Use the **DEFCTL** directive to define additional **CTL** values; however, the replacement only occurs if the original **CTL** code is rejected. Use the **DEFCTL WINDOW** format to maintain the definition for the current window only. If **WINDOW+** is used, the definition will cascade to lower level windows.

Format 1: *Define/Delete CTL Values*

DEFCTL [**WINDOW**] *eom\$*={*ctl-val* | *}

Use **DEFCTL** to define an additional **CTL** value to be returned for a given **EOM** (End-of-Message) value or to delete one. The **EOM** string is the sequence of characters received from the terminal to end the current input. The first character of the string must be a non-printable character between \$00\$ and \$1F\$, or \$7F\$; e.g.,

```
0010 DEFCTL $09$=6 ! Set TAB key to return CTL=6
0020 DEFCTL $1B38$=7 ! Set HOME <esc>H to CTL=7
0030 DEFCTL $09$=* ! Delete control value setting defined @line 0010
```

The normal setting for the TAB key is to return CTL = -1015.

```
0010 DEFCTL $09$ = -1015
```

Positive **CTL** values will be returned to the program. Negative **CTL** values have special significance to ProvideX. (Refer to the list of **Negative CTL Definitions**, at the end of this document.

Format 2: Redefine CTL Value

```
DEFCTL [WINDOW | WINDOW+] invalid_ctl={alternate | *}
```

Use this format of the **DEFCTL** directive to define alternate **CTL** values to be applied whenever the specified **CTL** value is received. For example, the **UP-ARROW** key is **CTL=-1011**. You can redefine it to return a **CTL=3**; e.g.,

```
0020 DEFCTL -1011=3 ! Return CTL 3 on up arrow
```

See Also

[CTL System Variable, p.557](#),
[CTL\(\) Function, p.410](#)
[Negative CTL Definitions, p.817](#)

Creating a Hot Key

ProvideX allows you to create a hot key that will **CALL** a program if it is pressed while the system is awaiting input.

To create such a hot key, define the control sequence that is generated by the key as a negative **CTL** value in the range -10 through -999. Then create and save your hot key program using the naming convention `$CTL-num` where *num* is the **CTL** value. Remember in your hot key program not to disturb the environment (i.e., do not close any files that are already open, but close any files you open in this program and remove any windows you create).

For example, the keystroke **Ctrl-A** returns `01`, which is defined in `MY_START_UP` program as a hot key:

```
1000 DEFCTL $01$=-500
```

ProvideX will call `$CTL-500` whenever anyone hits the hot key **Ctrl-A** throughout the current session.

During Conversions

When you convert BBx-style programs to ProvideX, it is sometimes necessary to have function and edit keys return single-character values. To do this, set both the 'EL' and 'FL' mnemonics and then use **DEFCTL** to redefine the mapped values and allow standard ProvideX functions to continue to operate. See also, '[EL' Mnemonic, p.604](#)', and the '[FL' Mnemonic, p.608](#)'.

Example

The following example redefines function keys **F5** through **F8** to return a single hex character (`$F5$` through `$F8$` respectively). It also maps some of the input edit keys to single character codes.

```
0100 DEFCTL $000074$=5 ! Reset F5 TO CTL=5
0100 ! ^ 100 - Map F5-F8 to return $F5$ through $F8$
0110 PRINT 'FL',"2"+CHR(4)+CHR(1)+$F5$, ! F5 = $F5$
0120 PRINT 'FL',"2"+CHR(5)+CHR(1)+$F6$, ! F6 = $F6$
0130 PRINT 'FL',"2"+CHR(6)+CHR(1)+$F7$, ! F7 = $F7$
0140 PRINT 'FL',"2"+CHR(7)+CHR(1)+$F8$, ! F8 = $F8$
0150 DEFCTL $F5$=5
0160 DEFCTL $F6$=6
0170 DEFCTL $F7$=7
0180 DEFCTL $F8$=8
0200 ! ^ 100 - Change Edit keys to single character
0210 PRINT 'EL',"2"+CHR(4)+CHR(1)+$01$, ! Home = $01$
0220 PRINT 'EL',"2"+CHR(5)+CHR(1)+$1A$, ! End = $1A$
0230 PRINT 'EL',"2"+CHR(6)+CHR(1)+$15$, ! PGUP = $15$
0240 PRINT 'EL',"2"+CHR(7)+CHR(1)+$06$, ! PGDN = $06$
0250 PRINT 'EL',"2"+CHR(8)+CHR(1)+$14$, ! Insert = $14$
0260 PRINT 'EL',"2"+CHR(9)+CHR(1)+$18$, ! Delete = $18$
0270 DEFCTL $01$=-1010 ! Home
0280 DEFCTL $1A$=-1018 ! End
0290 DEFCTL $15$=-1014 ! PGUP
0300 DEFCTL $06$=-1013 ! PGDN
0310 DEFCTL $14$=-1009 ! Insert
0320 DEFCTL $18$=-1007 ! Delete
```


DEFPRT Directive

Define as Printer

Format `DEFPRT (chan)col,ln`

Where:

chan Logical file number or channel of the file to define as a printer.

col Default maximum number of columns supported by the printer. This must be an integer value, range 0 to 255.

ln Default maximum number of lines supported by the printer. This must be an integer value, range 0 to 255.

Description Use the **DEFPRT** directive to specify that a given channel refers to a printer. You can specify the default maximum lines and columns supported by the printer. If you do not designate the file as a printer, ProvideX does not apply printer mnemonics, and the values for the **MXC()** and **MXL()** functions are not set.

You can use the **DEFPRT** directive for TCP files.

See Also [MNEMONIC Define File Command Sequence, p.210.](#)

Examples `0010 DEFPRT (LFO)80,25`

DEFTTY Directive

Define Terminal Size

Format DEFTTY [(*chan*)]*col*,*ln*

Where:

chan Channel or logical file number of the file to define as a terminal.

col Default maximum number of columns for the terminal. This must be an integer value, range 0 to 255.

ln Default maximum number of lines for the terminal. This must be an integer value, range 0 to 255.



Note: The number of lines has an additional limit based on a maximum of 20000 characters per window (i.e., the maximum number of lines is $20000/col$).

Description

Use the **DEFTTY** directive to designate that the logical file number refers to a terminal. You can specify the default maximum lines and columns supported by the terminal. If you do not designate the file as a terminal, Windows and other mnemonics will not be supported.

See Also

[MNEMONIC Define File Command Sequence, p.210.](#)

Examples

```
0010 DEFTTY (LFO)80,25
```

DELETE Directive *Remove Lines from Program*

Format `DELETE [start_stmtref][, [end_stmtref]]`

Where:

end_stmtref Last program statement to be deleted. Line label or number.

start_stmtref Starting program statement to be deleted. Line label or number.

Description Use the **DELETE** directive to delete a range of statements from the current program. For example, `DELETE SECT1 , 450` deletes from line label `SECT1` : to line `0450`, inclusive.

If you only include a starting statement reference, only that statement is deleted; e.g.,

```
DELETE 510 ! Deletes statement 0510
510 ! Is the same as DELETE 510
```

If you include both a starting reference and a comma, all statements from the reference to the end of the program are deleted. If you include a comma and an ending reference, statements from the start of the program up to and including the ending reference are deleted; e.g.,

```
DELETE 260, ! Deletes statements from line 0260 to program end
DELETE ,890 ! Deletes from program start, up to and including line number 0890
```

If you omit statement references, the complete program is deleted from memory.

Examples

```
->LIST
0010 REM "TEST"
0020 PRINT "TEST"
0030 STOP
->DELETE
->LIST
```

No program is in memory, so ProvideX doesn't return a listing and will report errors if you try to **EDIT** or **SAVE**.

DELETE OBJECT Directive *Remove Windows Object*

Format `DELETE OBJECT com_id[,ERR=stmtref]`

Where:

com_id Numeric variable to receive a handle (memory pointer to the COM object).

stmtref Program line number or label to transfer control to.

Description Use **DELETE OBJECT** to remove/disconnect associated Windows COM objects (OCX/OLE/ActiveX) from the screen.



Note: An OCX / OLE / ActiveX object is automatically destroyed when the window it was defined in is dropped.

Example:

```
0010 DEF OBJECT handle,@(2,2,70,16)="Shell.Explorer"
0020 errcode=handle'Navigate2("www.pvx.com")
0030 input *
0040 DELETE OBJECT handle
```

For information on the creation of an OCX (Object Component eXtension) object, see [DEF OBJECT Define Object, p.71](#).



Note: The **DELETE OBJECT** and **DROP OBJECT** directives can be used interchangeably regardless of the object being dropped. For further information, refer to the [DROP OBJECT Directive, p.104](#)

DICTIONARY Directive*Data Dictionary Access*

Restricted: Use of the **DICTIONARY** directive is *reserved exclusively for Sage Software Canada Ltd.* Its use is beyond the scope of this document and its syntax has been provided here only for the sake of completeness.

Formats

1. *Read:* **DICTIONARY READ**(*chan,fileopt*)*varlist*
2. *Write:* **DICTIONARY WRITE**(*chan,fileopt*)*varlist*
3. *Remove:* **DICTIONARY REMOVE**(*chan,fileopt*)

Where:

chan Channel or logical file number of file containing the data dictionary to access.

fileopt File options. Supported options for **DICTIONARY** include:

ERR=*stmtref* Error transfer

IND=*num* Field number

<0 for external key

>0 for internal fields

=0 file information record

varlist Comma-separated list of variables and/or literals.

Description

The **DICTIONARY** directive allows Sage Software Canada Ltd. to access and maintain the internal ProvideX data dictionary portions of the ProvideX databases.

DIM Directive

Define Arrays and Strings

Formats

1. *Define Array*: **DIM** *array_name*[\$](*subscript_1*[,*subscript_2*[,*subscript_3*]])
2. *Drop Array*: **DIM** *array_name*\$
3. *Define Composite String*: **DIM** *var*:\$**IOList**=*iolref*
4. *Initialize String*: **DIM** *var*\$(*len*[,*char*\$])[,...]

Where:

<i>array_name</i> [\$]	Numeric or string variable to be dimensioned as an array.
<i>char</i> \$	Value whose 1 st character will be used to fill the variable up to the length specified. String expression.
<i>iolref</i>	Either a string variable containing the object code of an IOList or a statement reference to an IOList (statement number or label).
<i>len</i>	Desired length of the string variable. Numeric expression, integer.
<i>subscript_1</i>	1 st dimensions (<i>min:max</i>) of array. Numeric expression, integers.
<i>subscript_2</i>	2 nd dimensions (<i>min:max</i>) of array. Numeric expression, integers.
<i>subscript_3</i>	3 rd dimensions (<i>min:max</i>) of array. Numeric expression, integers.
<i>var</i> \$	Name of string variable to be defined or initialized.

Description

Use the **DIM** directive to define an array, to define a composite string or to initialize a string. Refer to the **DIM() Function**, *p.415*, to read the total number, minimum number and maximum number of elements in an array. See also **System Limits**, *p.825*.



Note: For numeric arrays you can use either parenthesis () or square brackets [] to define arrays or reference elements. For string arrays, use square brackets [] to avoid confusion with substrings. All arrays have three subscripts. An array defined as *X*\$(4) has values 0 : 4 for subscript 1, 0 : 0 for subscript 2, and 0 : 0 for subscript 3. However, specifying the unused subscripts is unnecessary.

Format 1: Define Array

DIM *array_name*[\$](*subscript_1*[,*subscript_2*[,*subscript_3*]])

Use the **DIM** directive to define an array with one, two or three dimensions. Specify the dimensions using the values in the **DIM** statement. The dimensions of the array are defined as [*minimum*] : *maximum*. If you omit the minimum dimension, then ProvideX uses the default minimum, 0 zero. When you refer to items in arrays, the lowest subscript is the minimum value specified, the highest is the maximum value specified. You can redefine existing arrays using the **DIM** statement. The values of all elements in the array are automatically initialized to zero or null by the **DIM** statement.

The following example defines a three-dimensional array with three elements (0 through 2) in each dimension for a total of 27 elements:

```
DIM A(2,2,2)
```

The following example defines dimension one with 10 elements (1 through 10). Dimensions 2 and 3 are 0 : 0:

```
DIM CATS[1:10] !
```

Format 2: Drop Array Definition

DIM *array_name*[\$]

An array definition will be dropped by using the **DIM** directive without definitions/brackets; e.g., `DIM A$` or `DIM A`. Issuing `DIM A$[0]` does not remove the array definition from memory.

Format 3: Define Composite String

DIM *var*[:IOL=*iolref*]

Use this format of the **DIM** directive to define a composite string variable consisting of the fields specified in an IOList. The variable returns a string made up of the fields you specified in the IOList. If you modify the variable (e.g., by adding a field) ProvideX will modify the fields in the IOList. Field names in the IOList are prefixed by the name of the variable and a dot (.); e.g.,

```
0010 DIM CST$:IOL=1000
1000 IOLIST NAME$,ADDR$,CITY$,ZIP
```

yields the string `CST$` consisting of `CST.NAME$`, `CST.ADDR$`, `CST.CITY$`, and `CST.ZIP`. Each field is separated by the **SEP** or delimiter in the default record format.

You can use a **CLEAR** directive to reset a variable that is defined as a string template; e.g., `CLEAR CST$`. For more information refer to [Composite Strings vs BBx Templates, p.88](#), [IOLIST Specify Variable List, p.165](#), and [CLEAR Reset Variables, p.54](#).

The following example defines `PRD$` as a 36 character string consisting of `PRD.DESC$` for 30 characters and `PRD.COST` for 6 digits with a scale of 2 (2 implied decimal digits):

```
0010 DIM PRD$:IOL=2000
2000 IOLIST DESC$:[CHR(30)],COST:[NUM(6,2)]
```

Format 4: Initialize String With Fill Character

DIM *var*\$(*len*[,*char*])[,...]

Use the **DIM** directive to define a string variable of a specific length. You can also use the **DIM** statement to define the value of the string variable. ProvideX uses the first character of the value you set in the *char*\$ string as the fill character for the string being defined.

The fill character is repeated for the length of the string, as follows:

```
DIM A$(5, "*") is the same as DIM A$(5, "*-") ... both yield A$="*****"
```

If a value is not given (or is null) then the string is initialized to spaces.

Combine this string initialization with the definition of string arrays (above) to pre-initialize all the elements of a string array. For example, the **DIM** statement `DIM ACC_IDS$(10) (6, "0")` defines an array of 11 strings, each pre-initialized with 6 zeroes.

Composite Strings vs BBx Templates

ProvideX composite strings are made up of the variables you specify in the composite string definition, whereas string templates are strings which are parsed to obtain the various *logical* variables. This has the following impact, which you should consider when you design applications:

- You can pass a variable that is part of a composite string (e.g., `CST.NAME$`) to a subprogram and have a value returned in it. In a string template, however, `CST.NAME$` would be considered a substring which could not be altered by a subprogram.
- There is no performance impact when referencing individual variables in a composite string. There can be a performance impact when referencing logical variables in a template, especially when the template has variable length fields. On the other hand, when you reference the complete composite string, the string will be reconstructed each time. The string template always exists in memory, with no reconstruction required.
- Unlike string templates, composite strings can be reconstructed on the fly; e.g.,

```
0010 IOLIST NAME$, ADDR$, CITY$, AMT
0020 IOLIST NAME$, ADDR$, CITY$, ZIP$, AMT
0030 CST.ZIP$=" "
0040 DIM CST$:IOL=0010
0050 READ RECORD (1)CST$
0060 DIM CST$:IOL=0020
0070 WRITE RECORD (1)CST$
```

The logic above automatically inserts `CST.ZIP$` into the composite string while preserving the other data elements. Note that composite string variables are not implicitly cleared when the composite string is defined, whereas templates are.

- Since composite strings reference real variables, fields in composite strings have data types associated with them. That is, in a composite string, ProvideX considers `CST.AMT`, `CST.AMT$`, and `CST.AMT%` to be three different fields, whereas in a string template these would all refer to the same field.
- ProvideX does not currently support subscripting with composite strings.

DIRECT Directive Create File with Keyed Access

Format **DIRECT** *filename\$*,*max_len*[,*max_recs*[,*rec_size*]][,*ERR=stmtref*]

Where:

- filename\$* Filename of the **DIRECT** (Keyed) file. String expression. Mandatory.
- max_len* Maximum length of the key for all records in the file. Mandatory.
Numeric expression, integer.
- max_recs* Maximum number of records in the file. Optional numeric
expression. The default is zero (no limit). (Use a comma with no
value to set the default.)

If a positive value is supplied, ProvideX creates and pre-allocates
disk space for the file. With a negative value, ProvideX allocates
sufficient disk space for the file, but will set the *max_recs* count
back to zero (unlimited).
- rec_size* Maximum size of the data portion of each record (excluding the key).
Optional. Numeric expression. You can use:
No Value: Default is VLR with maximum size of 256.
Positive Integer: FLR of size specified.
Negative Integer: VLR with maximum length specified.
- stmtref* Program line number or label to transfer control to.

Description Use the **DIRECT** directive to create a Direct file with an external key field. ProvideX considers a Direct file to be the same as a Keyed file with an external key. If you use a filename that already exists, ProvideX returns an `Error #12: File does not exist (or already exists)`. The maximum size of the key to the file is mandatory along with the filename. The file type can be controlled by setting the '**KF**'= [System Parameter, p.671](#).

You can limit the number of records by specifying a maximum (an integer other than zero). If you do attempt to set a maximum, then attempt to exceed this value (e.g., on a **WRITE** statement) an `Error #2` is generated. You can use zero (0) to create a dynamic file, limited by physical file size limits and the amount of available drive space.

If you include the maximum data length, it must be long enough to hold the combined length of all the data fields and field separators for each record written to the file.

WindX supports the use of this directive via the [WDX] tag; e.g., `DIRECT "[WDX]somefile.ext" ...` See [\[WDX\] Direct Action to Client Machine, p.801](#).

See Also [CREATE TABLE Create Keyed File \(EFF\), p.58](#)
[KEYED Create Single/Multi-Keyed File, p.166](#)
[File Types, User's Guide](#)

Examples

```
0110 DIRECT A$+"-"+B$,10,100,50,ERR=1090
0200 DIRECT "CSTFLE",6,0,-128
```

Line 0200 creates a file with the following structure:

```
Keyed file: C:\Program Files\Sage Software\ProvideX\CST\CSTFLE
Maximum Record size .....: 128 (variable)
Maximum # records .....: (No limit)
Current # records .....: 0
Size of key block .....: 2048 bytes
External key size .....: 6
```

DIRECTORY Directive

Create Subdirectory

Format **DIRECTORY** *name*[\$[,**ERR**=*stmtref*]

Where:

name\$ String variable contains the name of the directory to create.

stmtref Program line number or label to transfer control to.

Description Use the **DIRECTORY** directive to create a directory in the operating system's file structure.

WindX supports the use of this directive via the [WDX] tag; e.g., **DIRECTORY** "[WDX]somefile.ext" ... For more information, see [\[WDX\] Direct Action to Client Machine, p.801](#).

Examples

```
0010 DIRECTORY "DATA"
0020 DIRECTORY "WRK." +STR(DEC($00$+GID))
0030 DIRECTORY "/tmp/WRK1"
0040 DIRECTORY "/tmp/WRK1/WRK2"
```



Note: A subdirectory can only be created within an existing directory; therefore, WRK1 must exist in order to create WRK2.

DISABLE Directive *Disable Use of Prefix Table Entry*

Format **DISABLE** (*prefix*[,**ERR**=*stmtref*])

Where:

prefix Numeric value of the prefix to disable. Numeric expression, integer.

stmtref Program line number or label to transfer control to.

Description Use the **DISABLE** directive to notify ProvideX that you do not want to use a given prefix table entry. This directive is used primarily in the conversion to or from old Business Basic languages where you could disable a specific disk drive.

To re-enable the prefix use the **ENABLE** directive.



Note: This directive is included in ProvideX for compatibility with other languages.

See Also **ENABLE Re-Enable Use of Prefix Table Entry**, *p.110*,
PREFIX Set File Search Rules, *p.249*.

Examples

```
->PREFIX (1) "/disk1/data/"  
->PREFIX (2) "/disk2/data/"  
->DISABLE (1)
```

Causes ProvideX to ignore PREFIX (1).

DISABLE CONTROL Directive

Disable Control

- Formats**
1. *Disable Single Control*: **DISABLE CONTROL** *ctl_id1[:sub_id][,ctl_id2[:sub_id]...][,ERR=stmtref]*
 2. *Disable Multiple Controls*: **DISABLE CONTROL** *bin_list\$[,ERR=stmtref]*

Where:

bin_list\$ One or more three-byte binary strings identifying controls:
 BIN(control_ID, 2)+\$00\$ for most controls or
 BIN(control_ID, 2)+bin(sub_id, 1) for radio buttons

ctl_id Value of the control(s) to disable. Numeric expression, integer. If you include a list, use the comma as the separator.

stmtref Program line number or label to transfer control to.

sub_id Unique radio button ID. Numeric expression (range 1 to 254).

Description Use the **DISABLE CONTROL** directive to notify ProvideX that you want the specified control (button, check box, radio button, etc.) disabled. The *bin_list\$* expression will support up to 30 controls (90 characters).

To re-enable the control use the **ENABLE CONTROL** directive.

Example

```
00010 print 'CS'
00020 button 10,@(1,1,20,2)="Show Message"
00030 obtain x
00040 if ctl=10 then msgbox "Hello World","Button Message"
00050 if ctl=4 then stop
00060 if ctl=1 then enable control 10; print @(24,1),"Enabled "
00070 if ctl=2 then disable control 10; print @(24,1),"Disabled"
00080 goto 0030
```

See Also [ENABLE CONTROL Directive, p.111.](#)

DISABLE EVENT Directive

Internal Event Disable

Formats

1. *Timer Event*: **DISABLE EVENT ON TIM**
2. *Data Available on Channel Event*: **DISABLE EVENT ON DATA** (*chan*)
3. *On Close of Channel Event*: **DISABLE EVENT ON CLOSE** (*chan*)
4. *On Open Event*: **DISABLE EVENT ON OPEN**
5. *On Class Load Event*: **DISABLE EVENT ON LOAD CLASS**

Where:

chan Channel or logical file number.

Description

Use the **DISABLE EVENT** directive to disable the handling of various system events within a ProvideX session. The generation and trapping of events requires that the internal "`*system.pvc`" COM object be defined first.

See Also

[ENABLE EVENT Directive, p.112.](#)

Example

```
00010 DEF OBJECT PvxComID,"*system" ! Activate support for System Events
00020 ENABLE EVENT ON TIM=2 ! Activate timer event at two second
      intervals
00030 ON EVENT "TimeOut" FROM PvxComID PREINPUT 99 ! Establish how to
      handle the event
00031 c=0
00040 WHILE 1
00050 INPUT *
00060 IF CTL=99 \
      THEN c++;
          PRINT "Timeout ",c;
          IF c>3 \
            THEN BREAK
00070 WEND
00075 DISABLE EVENT ON TIM ! Deactivate timer event
00077 DROP OBJECT PvxComID
00080 END
```

DROP Directive *Removes Program from Memory*

Format `DROP prog$[,ERR=stmtref]`

Where:

prog\$ Name of the program to be unloaded from memory. String expression.

stmtref Program line number or label to transfer control to.

Description Use the **DROP** directive to unload a program previously loaded into memory by the **ADDR** directive. The memory used while the program was loaded is returned to the system. ProvideX returns an Error #17: Invalid file type or contents if you use a **DROP** statement for a program that is not currently loaded in memory.



Note: In *Object Oriented Programming*, **DROP OBJECT** can be used to delete an object and **DROP CLASS** can be used to delete a class. See [DELETE OBJECT](#) and [DELETE OBJECT](#) for more information.

See Also [ADDR Load & Lock Program in Memory, p.30](#)
[Data Integration, User's Guide.](#)

Examples `-->DROP "ARDATE"`

DROP_BOX Directive

Control Drop Box


Formats

1. *Define/Create*: **DROP_BOX** *ctl_id*,@(col,ln,wth,ht)[,ctrlpt]
2. *Remove*: **DROP_BOX REMOVE** *ctl_id*[,ERR=stmtref]
3. *Disable/Enable*: **DROP_BOX** {DISABLE | ENABLE}*ctl_id*[,ERR=stmtref]
4. *Hide/Show*: **DROP_BOX** {HIDE | SHOW} *ctl_id*[,ERR=stmtref]
5. *Force Focus*: **DROP_BOX GOTO** *ctl_id*[,ERR=stmtref]
6. *Signal on Focus*: **DROP_BOX SET_FOCUS** *ctl_id*,*ctl_val*[,ERR=stmtref]
7. *Load via Delimited String*: **DROP_BOX LOAD** *ctl_id*,*dlim_list*[,ERR=stmtref]
8. *Load via Array*: **DROP_BOX LOAD** *ctl_id*,*array_name*{**ALL**}[,ERR=stmtref]
Note: The curly braces enclosing **{ALL}** are part of the syntax.
9. *Load/Delete Index Element*: **DROP_BOX LOAD** *ctl_id*,*index*,{*element*\$ | *}[,ERR=stmtref]
10. *Retrieve Element*: **DROP_BOX FIND** *ctl_id*,*index*,*var*[,ERR=stmtref]
11. *Read Current String*: **DROP_BOX READ** *ctl_id*,*var*[,*mode*][,ERR=stmtref]
12. *Read Current Index*: **DROP_BOX READ** *ctl_id*,*var*[,*mode*][,ERR=stmtref]
13. *Reset Using Selection*: **DROP_BOX WRITE** *ctl_id*,*element*[,ERR=stmtref]
14. *Reset Using Index*: **DROP_BOX WRITE** *ctl_id*,*index*[,ERR=stmtref]
15. *Clear Current Selection*: **DROP_BOX WRITE** *ctl_id*, "",[,ERR=stmtref]
16. *Report All Changes*: **DROP_BOX AUTO** *ctl_id*[,ERR=stmtref]

Where:

@(col,ln,wth,ht) Position and size of the drop box region when expanded. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines. (Note that drop box height, when not expanded, is governed by the system and is roughly 1.5 times the standard graphic font height.)

array_name\$ Name of array to load into drop box. String variable followed by **{ALL}**.

ctl_id Unique logical identifier for the drop box (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the  key) or **Negative CTL Definitions, p.817**. Use this value with the apostrophe operator to access various **Drop Box Properties**.

ctl_val CTL value to generate when the drop box gains focus.

<i>ctrlopt</i>	<p>Control options. Supported options for DROP_BOX include:</p> <p>ERR=stmtrf Error transfer</p> <p>FNT="font,size[,attr]" Font name, size, optional properties. Refer to the 'FONT' Mnemonic, p.609 for details.</p> <p>KEY=char\$ Hot key</p> <p>MSG=text\$ Message line</p> <p>MNU=ctl CTL value associated with right-click menu event.</p> <p>TBL=char\$ Single character translation</p> <p>TIP=text\$ Mouse pointer message.</p> <p>To change the colour, refer to the 'TC'= System Parameter, p.689.</p> <p>OWN=name\$ Name assigned for automated testing of this control.</p> <p>OPT=char\$ Attribute/behaviour settings:</p> <ul style="list-style-type: none"> "A" - <i>Auto</i>. Generate CTL signal when a new element is highlighted. "B" - <i>No border</i>. Drop box will not have a border. "d" - <i>Permanently Disabled</i>. Drop box cannot be enabled. "D" - <i>Initially Disabled</i>. User cannot access the drop box. "h" - <i>Permanently Hidden</i>. Drop box cannot be shown. "H" - <i>Initially Hidden</i>. Drop box is initially hidden. "G" - <i>Global</i>. Keep active on focus change to new/non-concurrent window. "S" - <i>Signal</i>. Generate CTL value but without shifting focus. "s" - <i>Scroll</i>. Allow scroll within resizable/scrollable dialogue box. "T" - <i>Strip trailing spaces</i>. "X" - <i>Signal when focus exits from control</i>. "Z" - <i>Cursor changes to "resize" pointer if within 4 pixels of control</i>. <p>Some characters may be combined. Invalid settings are ignored.</p>
<i>dln_list\$</i>	Delimited list of elements to load. String expressions.
<i>element\$</i>	Single element to load. String expression. You can use the <i>asterisk</i> * instead to <i>delete</i> an element. For instance, DROP_BOX LOAD 86, 4, * will remove element 4 from the DROP_BOX .
<i>index</i>	Position of the element in the drop box. Numeric expression. Integers: the index of the 1 st element is 1.
<i>mode\$</i>	String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to activate the drop box (\$01\$ for MOUSE-CLICK or \$0D\$ for Enter).
<i>var[\$]</i>	Variable to receive value. String variable for element/numeric for index.

Description Use the **DROP_BOX** directive to create and manipulate drop box control objects on the screen. A drop box normally displays a single line on the screen with a **DOWN-ARROW** on the right side. The user can select any element from a list of items you assign to the drop box, but variable input is *not* allowed. That is, the user can only select, *not enter*, values. To view the list, the user clicks on the **DOWN-ARROW**. When the user selects a drop box item, the associated *ctl_id* is used to generate a CTL value.

Refer to [VARDROP_BOX Control Variable Drop Box, p.354](#), if you need a drop box that allows both variable input and selection from a list.

Because a drop box list is in drop down form, a drop box takes a smaller amount of space on the screen than a comparable list box. In addition, ProvideX automatically supplies vertical scrollbars if the number of elements overflows the drop-down box size. Combine these features to optimize the screen design when display space is at a premium.

Drop Box Properties

The [Apostrophe Operator](#) can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a drop boxes are described in [Chapter 7. Control Object Properties, p.704](#).

Format 1: Define/Create

DROP_BOX *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]

Use this format to define or create a drop box and give it a unique identifier in *ctl_id*. You can use the **FNT=** option to establish the font for it. If you omit the font option, ProvideX uses the system default font. Use **FNT="*"** to set the font as standard text mode fixed font.

The following example creates a drop box that generates a CTL=100 when any item is selected from it. It is loaded with the items Cat, Dog, and Pig.

```
0010 DROP_BOX 100 ,@(2,14,12,6)
0020 DROP_BOX LOAD 100 , "Cat/Dog/Pig/"
```

Format 2: Remove

DROP_BOX REMOVE *ctl_id*[,ERR=*stmtref*]

Use the **DROP_BOX REMOVE** format to delete a drop box.

Format 3: Disable/Enable

DROP_BOX {**DISABLE** | **ENABLE**}*ctl_id*[,ERR=*stmtref*]

Use **DROP_BOX DISABLE** to gray out a drop box so that it will be visible but *inaccessible*, to users. To reactivate it, use **DROP_BOX ENABLE**.

Format 4: Hide/Show

DROP_BOX {**HIDE** | **SHOW**} *ctl_id*[,ERR=*stmtref*]

With the **DROP_BOX HIDE** format, the drop box remains active, but is not displayed. It is still accessible programmatically. Use the **DROP_BOX SHOW** format to restore the display and user access.

Format 5: *Force Focus*

```
DROP_BOX GOTO ctl_id [,ERR=stmtref]
```

Use the **DROP_BOX GOTO** format to reactivate and force focus to a drop box, ready for the next user action.

Format 6: *Signal on Focus*

```
DROP_BOX SET_FOCUS ctl_id, ctl_val [,ERR=stmtref]
```

Use the **DROP_BOX SET_FOCUS** format to define an alternate CTL value to generate whenever focus shifts to the drop box.

Formats 7, 8 and 9: *Load a Drop Box*

Use **DROP_BOX LOAD** to add or delete the elements listed in a drop box. To add elements, you can load them from a *delimited string*, as an *array of string elements*, or *individually*. If you load more items into the list than the drop down box can display at one time (because of its physical size), ProvideX automatically supplies scrollbars.

```
DROP_BOX LOAD ctl_id,dlm_list$ [,ERR=stmtref]
```

Load List. When you load a drop box from a *delimited string*, the last character in the string must be the delimiter. That delimiter must be identical to the separator between the elements in the string. For example, in the delimited string below, the delimiter is the slash "/". It ends the string and separates the elements; e.g.,

```
DROP_BOX LOAD 12000, "Fox/Cat/Dog/Cow/Sheep/Horse/Pig/Elephant/Ant/"
```

To *clear* all elements from a drop box, use a null string; e.g.,

```
DROP_BOX LOAD 86, ""
```

```
DROP_BOX LOAD ctl_id,array_name ${ALL} [,ERR=stmtref]
```

Load Array. Use this format to load a complete *array* into the drop box. Note that the curly braces enclosing **{ALL}** are part of the syntax; e.g.,

```
DROP_BOX LOAD 16000, A${ALL}.
```

```
DROP_BOX LOAD ctl_id,index, {element$ | *}, [ERR=stmtref]
```

Load Element. When you load *individual* drop box elements, use an integer value to state the index of the element before which to insert the element being loaded. For instance, if *index* is 1, the new element will be inserted before 1, at the start of the list. If *index* is 0 *zero*, the new element will be appended to the end of the list.

To *delete* or *remove* a specified element from a drop box, use an *asterisk* * in place of the element string; e.g.,

```
DROP_BOX LOAD 86,4,* ! Deletes list element 4 from a box whose ctl_id is 86.
```

Format 10: *Retrieve Element*

DROP_BOX FIND *ctl_id,index,var\$[,ERR=stmtref]*

Use the string variable in **DROP_BOX FIND** to retrieve a specific element from a drop box.

Formats 11 and 12: *Read Current Selection*

Use the **DROP_BOX READ** formats to read which element the user has selected from the drop box. You must read the user's selection before your application can use it.

Use the (optional) *mode\$* variable to have ProvideX return the user's method of selecting an item from the drop box.

Some possible return values are:

\$01\$ for **MOUSE-CLICK**.

\$0D\$ for **Enter**.

\$00\$ when the user **Tab** 's away from the drop box.

\$00\$ when the user exits the control.

After this value is read, the value in *mode\$* is reset to \$00\$ (null).

DROP_BOX READ *ctl_id,var\$[,mode\$][,ERR=stmtref]*

Read Current Element. With this format, ProvideX returns the string contents of the user's currently selected element in *var\$* and, optionally, the user's method of selection in *mode\$*.

DROP_BOX READ *ctl_id,var[,mode\$][,ERR=stmtref]*

Read Current Index. Use **DROP_BOX READ** to read the index of the user's current selection and (optionally) the *mode\$* of selection.

Formats 13 and 14: *Write Current Selection*

Use the **DROP_BOX WRITE** formats described below to reset the items selected from a drop box.

DROP_BOX WRITE *ctl_id,element\$[,ERR=stmtref]*

Reset Using Selection. Resets the currently selected item to reflect the specified element. If it does not exist, ProvideX returns an Error #11: Record not found or Duplicate key on write.

DROP_BOX WRITE *ctl_id,index[,ERR=stmtref]*

Reset Using Index. Resets the current item to the specified index, which is the same same as setting the '**CurrentItem**' property. If it does not exist, ProvideX returns an Error #11: Record not found or Duplicate key on write.

Format 15: *Clear Current Selection*

DROP_BOX WRITE *ctl_id*,'" [,ERR=*stmtref*]

Use this format to clear the currently selected entry in drop boxes.



Note: This behavior can be altered by use of the '+N' & '-N' Mnemonics, p.623.

Format 16: *Report All Changes*

DROP_BOX AUTO *ctl_id* [,ERR=*stmtref*]

Use the **DROP_BOX AUTO** format to have ProvideX generate a CTL value to report all changes whenever the user highlights an element from a drop box list.

Example

```

0010 ! DROP_BOX Creation and Use
0020 PRINT 'CS'; LIST
0030 DROP_BOX 88,@(30,18,15,10)
0040 DROP_BOX LOAD 88,"CAT,DOG,PIG,FOX,"
0050 DROP_BOX WRITE 88,"CAT"
0060 LET D_BX=88
0070 SETCTL D_BX:READ_BOX
0080 DROP_BOX GOTO D_BX
0090 PRINT @(30,24),"Try Mouse and/or keyboard to select. END=<F4>"
0100 OBTAIN (0,SIZ=1,ERR=0100)@(0,0),'CURSOR'("off"),'ME',IN_VAR$,'MN'
0110 LET CT=CTL; IF CT=4 THEN GOTO END
0120 READ_BOX:
0130 DROP_BOX READ D_BX,ANIMAL$,STROKE$,ERR=0001
0140 DROP_BOX READ D_BX,IND_NUM
0150 PRINT @(30,20),"HTA(SELECTION) : ",HTA(STROKE$)," AND CTL=",CTL:"#####"
0160 PRINT @(30,21),"Index for ",ANIMAL$,@(44,21)," : ",IND_NUM,""
0170 GOTO 0100
0180 END:
0190 DROP_BOX REMOVE D_BX
0200 PRINT 'CS'

```

DROP CLASS Directive

Delete Class Definition

Format **DROP CLASS** *class*\$

Where:

class\$ Name of the class to be deleted. String expression.

Description The **DROP CLASS** directive is used in *Object Oriented Programming* to delete a class definition and all related information. Once a class definition is established, then it may not be changed but it can be deleted and recreated using **DROP CLASS**.

Only class definitions that have no references to them may be deleted. This means that no class can be deleted when:

- Any object exists which refers to this class.
- Any other class exists which refers to this class by its **LIKE** clause.

Any attempt to delete a class that has a reference to it will return an Error #50: "Class in use or already defined".

All class definitions will be deleted when a **START** directive is issued. Either **DROP CLASS** or **DELETE CLASS** can be used to delete a class definition.

See Also

[DEF CLASS Define Object Class, p.65](#)

[DROP OBJECT Delete Object, p.104](#)

[LOAD CLASS Pre-Load Class Definition, p.195](#)

[RENAME CLASS Change Name of Class, p.283](#)

[STATIC Add Local Properties at Runtime, p.329](#)

[NEW\(\) Function, p.489](#)

[REF\(\) Function, p.512](#)

[Data Integration, User's Guide](#)

DROP INDEX Directive *Drop Key from Keyed File*

Format **DROP INDEX** {*keynumber* | *keyname*\$} **FROM** *filename*\$ [,**ERR**=*stmtref*]

Where:

filename\$ Name of the file from which the key will be dropped. String expression.

FROM Mandatory keyword, not case-sensitive.

keyname\$ Name of the key to drop (if assigned). String expression.

keynumber Key number (KNO value) to drop.

stmtref Program line number or label to transfer control to.

Description The **DROP INDEX** directive drops keys from a ProvideX Keyed file without having to rebuild the file. When dropping keys from a file:

- The key is removed from the data file and the space previously occupied by the key table is made available for subsequent use within the file.
- Only one drop can be processed against a file at one time.
- Exclusive access to the file is required.
- The primary key is required and cannot be dropped.

See also [ADD INDEX Add Key to Keyed File, p.29](#)
[RENAME..INDEX Rename Keys in Keyed File, p.285](#)

Example

```
DROP INDEX 3 FROM "cstfile"  
DROP INDEX CustName FROM "cstfile"
```

DROP OBJECT Directive

Delete Object

Format `DROP OBJECT obj_id[,ERR=stmtref]`

Where:

obj_id Object Identifier

stmtref Program line number or label to transfer control to.

Description The **DROP OBJECT** directive is used in *Object Oriented Programming* to delete an object. Only objects whose reference count is 1 can be deleted. Once an object is destroyed, its identifier may be re-assigned to another object; although, this is not recommended.

Objects are destroyed when the application issues a **START** directive.

REF (READ obj_id) returns the current reference count value. All calls to **REF ()** return the current reference count (or 0, if deleted).



Note: The **DELETE OBJECT** and **DROP OBJECT** directives can be used interchangeably regardless of the object being dropped. For further information, refer to the **DELETE OBJECT Directive, p.84**

See Also

DEF CLASS Define Object Class, *p.65*

DROP CLASS Delete Class Definition, *p.102*

LOAD CLASS Pre-Load Class Definition, *p.195*

RENAME CLASS Change Name of Class, *p.283*

STATIC Add Local Properties at Runtime, *p.329*

NEW() Function, *p.489*

REF() Function, *p.512*

Data Integration, User's Guide


DROP..ON Directive

Drag and Drop

Format

1. *Define Drag & Drop*: **DROP** *source_ctl_id* **ON** *dest_ctl_id* **RETURN** *new_ctl_id*
2. *Remove Logic*: **DROP** *source_ctl_id* **ON** *dest_ctl_id* **REMOVE**
3. *Temporarily Disable*: **DROP** *source_ctl_id* **ON** *dest_ctl_id* **DISABLE**
4. *Re-Enable Drag & Drop*: **DROP** *source_ctl_id* **ON** *dest_ctl_id* **ENABLE**

Where:

- dest_ctl_id* Unique numeric identifier for the destination control.
- new_ctl_id* The CTL signal that is generated when the drop occurs on the destination object. Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the  key) or special negative CTL values set by the system. See [Negative CTL Definitions, p.817](#).
- source_ctl_id* Unique numeric identifier for the source control.

Description

The Drag and Drop feature allows for copying or moving of information from one object to another object. The drop event is triggered when a mouse is used to select information from the source object, drag the selection to another object, and deposit the information at the new location. The application is responsible for adding the information to the destination object and removing it from the source object.

The available *Drag From* or source controls are multi-lines and various types of list boxes, drop boxes, and grids.

The available *Drop On* or destination controls are buttons, check boxes, drop boxes, grids, multi-lines, list boxes, radio buttons, tristate boxes. The highlighting of items on which the drop will occur is not supported in simple or formatted list boxes; however, this is supported in list view, report view, and tree view objects.



Note: When using the **DISABLE** syntax, the cursor will still change indicating that the drop is valid, however the signal indicating the drop took place is not generated.

The following illustrates the use of **DUMP OBJECT**:

```
->DUMP OBJECT 100015
! Dump of OBJECT information
_Obj=100015
_Class$="*rpt/RptSort"
_Refcnt=1
! External Properties
Order$="A"
Length=6
Type$="S"
SegmentName$="AccountID"
```



EDIT Directive

Edit Line in Program

Formats

1. *Edit*: **EDIT** *stmtref*{**D**[string] | **C**[string] | **R**[string] | [string]}

2. *Edit using back apostrophe* : ` *stmtref*{**D**[string] | **C**[string] | **R**[string] | [string]}

Note: The square brackets above are part of the statement's syntax.

Where:

` *Back Apostrophe*. ProvideX accepts this as a substitute for typing EDIT.

[string] *Add*. String literal to add to the statement.

C[string] *Copy*. String literal contains the final character(s) of string to be copied.

D[string] *Delete*. String literal contains the final character(s) of the string to be deleted.

R[string] *Replace*. String literal is the new text to replace the existing string.

stmtref Program line number or label to transfer control to.



Note: The **EDIT** directive is used in Command mode.

Description

Use the **EDIT** directive to change an existing statement in a program. ProvideX builds a new statement based on the commands you use in the **EDIT** directive. The new statement then replaces the existing one, unless you change the line number. (In the latter case, the original statement remains in the program at the original line number, and the new or resultant statement is added to the program at the new line number.)

If you do not include parameters following the statement reference, ProvideX displays the specified line so that you can edit it directly using the keyboard.

Four options are available in the EDIT directive.

Delete text from statement. Use the **D** option to scan along the original statement up to and including the given string without copying the data to the new statement. The result omits the deleted portion of the string from the new statement.

Copy text from current statement to the new statement. Use the **C** option to scan along the original statement from the cursor up to and including the given string, copying the data to the new statement.

Replace text from current line in the new statement. Use the **R** option to append the given string to the new statement while advancing over the corresponding number of characters in the original statement.

Add text to the new statement. This option adds the given string to the new statement.

After all options have been processed, any remaining portion of the original statement is appended to the new statement.

Examples

```
0200 LET A=4*M; PRINT "ANSWER =",A  
EDIT 200 C[=] D[*] C[A] R[nswer] [ now]
```

After **EDIT**:

```
0200 LET A=M; PRINT "Answer now =",A
```

ENABLE Directive *Re-Enable Use of Prefix Table Entry*

Format **ENABLE** (*prefix*[,**ERR**=*stmtref*])

Where:

prefix Numeric value of the **PREFIX** to re-enable. Numeric expression.

stmtref Program line number or label to transfer control to.



Note: This directive is used primarily in the conversion to or from old Business Basic languages where you could **DISABLE** and **ENABLE** a specific disk drive.

Description Use the **ENABLE** directive to notify ProvideX that you want a given prefix reactivated. To disable a prefix use the **DISABLE** directive.

See Also [DISABLE Disable Use of Prefix Table Entry, p.92](#),
[PREFIX Set File Search Rules, p.249](#).

Example

```
0100 PREFIX (1) "/disk1/data/"
0500 DISABLE (1) ! Tell ProvideX to ignore PREFIX (1)
1000 ENABLE (1) ! Reactivate PREFIX (1)
```

ENABLE CONTROL Directive

Enable Control

Format

1. *Enable Single Control*: **ENABLE CONTROL** *ctl_id1*[:*sub_id1*][,*ctl_id2*[:*sub_id2*]...][,**ERR=***stmtref*]
2. *Enable Multiple Controls*: **ENABLE CONTROL** *bin_list*\$[,**ERR=***stmtref*]

Where:

bin_list\$ One or more three-byte binary strings identifying controls:
 BIN(*control_ID*, 2)+\$00\$ for most controls or
 BIN(*control_ID*, 2)+bin(*sub_id*, 1) for radio buttons

ctl_id Value of the control(s) to enable. Numeric expression, integer. If you include a list, use the comma as the separator.

stmtref Program line number or label to transfer control to.

sub_id Unique radio button ID. Numeric expression (range 1 to 254).

Description Use the **ENABLE CONTROL** directive to notify ProvideX to reactivate the specified control (button, check box, etc.). To disable the control use the **DISABLE CONTROL** directive.

See Also [DISABLE CONTROL Directive, p.93.](#)

Example

```
00010 print 'CS'
00020 button 10,@(1,1,20,2)="Show Message"
00030 obtain x
00040 if ctl=10 then msgbox "Hello World","Button Message"
00050 if ctl=4 then stop
00060 if ctl=1 then enable control 10; print @(24,1),"Enabled "
00070 if ctl=2 then disable control 10; print @(24,1),"Disabled"
00080 goto 0030
```

ENABLE EVENT Directive

Internal Event Enable

Formats

1. *Timer Event*: **ENABLE EVENT ON TIM**
2. *Data Available on Channel Event*: **ENABLE EVENT ON DATA** (*chan*)
3. *On Close of Channel Event*: **ENABLE EVENT ON CLOSE** (*chan*)
4. *On Open Event*: **ENABLE EVENT ON OPEN**
5. *On Class Load Event*: **ENABLE EVENT ON LOAD CLASS**

Where:

chan Channel or logical file number.

Description

Use the **ENABLE EVENT** directive to enable the handling of various system events within a ProvideX session. The generation and trapping of events requires that the internal "`*system.pvc`" COM object be defined first.

See Also

[DISABLE EVENT Directive, p.94](#)
[WAIT FOR EVENT Directive, p.373](#)

Example

```
00010 DEF OBJECT PvxComID,"*system" ! Activate support for System Events
00020 ENABLE EVENT ON TIM=2 ! Activate timer event at two second
      intervals
00030 ON EVENT "TimeOut" FROM PvxComID PREINPUT 99 ! Establish how to
      handle the event
00031 c=0
00040 WHILE 1
00050 INPUT *
00060 IF CTL=99 \
      THEN c++;
          PRINT "Timeout ",c;
          IF c>3 \
          THEN BREAK
00070 WEND
00075 DISABLE EVENT ON TIM ! Deactivate timer event
00077 DROP OBJECT PvxComID
00080 END
```


END Directive

Halt Program Execution

Format **END**

Description Use the **END** directive to halt the currently running program. If the current program is a subprogram, then control is immediately passed back to the calling program. Otherwise all open files are closed, a **RESET** operation is performed, and the next location counter is set to the start of the program.

If an application is invoked directly by an operating system command that specifies a lead program, then the **END** directive performs the function of a **QUIT** and automatically returns the user to the operating system. If the application is **RUN** from Command mode, ProvideX returns to Command mode.

When you use the **END** directive in a compound statement, it must be the final directive. (*Exception:* A remark can follow the **END** directive.)

The *END label emulates an **END** directive for use as a statement reference. The **END** directive is functionally identical to the **STOP** directive.

See Also [QUIT Terminate ProvideX, p.264](#),
[RELEASE Terminate ProvideX, p.279](#),
[STOP Halt Program Execution, p.330](#),
[Labels/Logical Statement References , p.816](#)
[Called Procedures, User's Guide](#)

END DEF Directive *End Definition of Multi-line Function*

Format **END DEF**

Description Use the **END DEF** directive to mark the end of a multi-line function or **DEF CLASS** statement. In Execution mode, when ProvideX encounters a **DEF FN** directive for a multi-line function, it skips forward until it encounters an **END DEF** directive. (That is, control is transferred to the line or statement after the end definition.) If you use an **END DEF** directive without a preceding **DEF FN** directive, ProvideX returns an Error #45: Referenced statement invalid.

It is also used to mark the conclusion of **Class Definition** (**DEF CLASS** statement) in **Object Oriented Programming**, p.22.

See Also **RETURN Subroutine/Function Return**, p.291
ESCAPE Interrupt Program Execution, p.122
DEF FN Define Function, p.68.
DEF CLASS Define Object Class, p.65.
Data Integration, *User's Guide*.

END SWITCH Directive *End Branching of a Program*

Format **END SWITCH**

Description Use the **END SWITCH** directive to stop the branching that has been activated in an application by a **SWITCH** directive.



Note: For complete syntax, refer to [SWITCH..CASE Branch Control, p.331](#).

See Also [SWITCH..CASE Branch Control, p.331](#),
[BREAK Immediate Exit of Loop, p.33](#),
[CASE Define Branch Points, p.42](#),
[DEFAULT Branch If No Matching Case, p.77](#).

Examples

```

00100 PROCESS_TAXCODE:
00110  LiquorTax=0,SalesTax=0,ServiceTax=0
00120  SWITCH UCS(TaxCode$)
00130  CASE "X","Z" ! two codes are tax exempt
00140  BREAK ! stop processing for case "X" here
00150  CASE "L" ! liquor pays all liquor,sales and service tax
00160  LiquorTax=cost*LiquorTaxRate
00170  ! no break here, logic falls through
00180  CASE "S" ! pays sales and service tax
00190  SalesTax=cost*SalesTaxRate
00200  ! no break here, logic falls through
00210  CASE "V" ! service tax
00220  ServiceTax=cost*ServiceTaxRate
00230  BREAK ! end processing for this case and any that fell through
00240  DEFAULT ! enter here if case not found
00250  MSGBOX "Unknown tax code","Error"
00260  END SWITCH
00270  TotalTax=LiquorTax+SalesTax+ServiceTax
00280  RETURN

```

END WITH Directive

End Branching of a Program

Format **END WITH**

Description Use the **END WITH** directive to signal the end of a **WITH** construct.



Note: For complete syntax, refer to [WITH Object Reference Construct, p.382](#).

See Also [WITH Object Reference Construct, p.382](#)

END_IF Directive

End IF Directive

Formats

1. *End IF Before Line Ends*: **IF expression THEN ... ELSE ... END_IF ...**
2. *End IF using FI*: **IF expression THEN ... ELSE ... FI ...**

Where:

expression Condition to control processing.

... Directives, processing.



Note: **FI** is an accepted substitute for **END_IF**. Refer to the **IF.THEN..ELSE** directive for complete syntax.

Description

Use **END_IF** (or **FI**) to terminate an **IF** directive before the end of a statement or line. When an **END_IF** directive follows an **IF** directive, execution resumes immediately after the **END_IF** directive, regardless of whether the condition was found to be *true* or *false*.

See Also

[IF..THEN..ELSE Test Condition, p.157.](#)

Example

```
00100 PRINT "The customer has ",
00200 IF bal<=0 \
      THEN PRINT "NO", \
      ELSE PRINT "$",bal, \
      END_IF ;
      PRINT " credit available"
->BAL=0
->RUN
The customer has NO credit available.

->BAL=1.98
->RUN
The customer has $ 1.98 credit available
```

ENDTRACE Directive

End Trace Output

- Formats**
1. *End Trace:* **ENDTRACE**
 2. *End Tracing of Windows Host Program:* **ENDTRACE SERVER**

Where

SERVER *For internal use only.*

Description Use the **ENDTRACE** directive to stop the tracing of program statements (activated by a previous **SETTRACE** directive).

See Also [SETTRACE Enable Program Tracing, p.324](#)

Examples

```
0010 SETTRACE
0020 DEFCTL -1011=3 ! Return CTL 3 on up arrow
0030 INPUT "Enter name:",N$
0040 ON CTL GOTO 0050,0030,0030,0060,0060
0050 INPUT "Enter Addr:",A$
0060 ENDTRACE ; PRINT "DONE"; END
```

-:run

```
0010 SETTRACE
0020 DEFCTL -1011=3 ! Return CTL 3 on up arrow
0030 INPUT "Enter name:",N$
Enter name: ! User hit up arrow
0040 ON CTL GOTO 0050,0030,0030,0060,0060
0060 ENDTRACE ; PRINT "DONE"; END
DONE
```

ENTER Directive

Specify Arguments

Format **ENTER** [*arglist*][,**ERR**=*stmtref*]

Where:

- arglist* Variables in to receive arguments passed from the calling program. Use:
- a comma-separated list of simple numeric and/or string variables, not subscripts/substrings,
 - **IOL**=*ioltref* (e.g., **ENTER IOL**=8000), or
 - a complete numeric array (e.g., **ENTER ARRAY_NAME**{**ALL**})
- stmtref* Program line number or label to transfer control to.



Restrictions: You can only use this directive in *called* programs (subprograms).

Description Use **ENTER** in a called program to define the total number, relative positions and types of variables it will receive. These are arguments passed to the subprogram via the calling program's **CALL** statement.

The variables in the calling program's **CALL** statement must match those in the subprogram **ENTER** statement exactly. That is, each argument in the **CALL** statement must correspond by position and in type (numeric or string) to a variable in the **ENTER** statement. Otherwise, ProvideX returns Error #36: **ENTER** parameters don't match those of the **CALL**.

If the calling program is passing a complete numeric array, the name of the array must be specified, followed by {**ALL**} in both the **ENTER** and **CALL** statements (curly brackets are part of the syntax).

Where a **CALL** statement specifies a simple variable, all changes made to the variable **ENTER**ed in your subprogram will be reflected in the calling program when the subprogram terminates. You can protect a simple variable in either the **CALL** or **ENTER** statement by placing the argument inside parentheses –this turns the variable into an expression, which has the effect of making it *read only*.

String templates cannot be passed if they are defined prior to the **ENTER** statement in the called program.

See Also [CALL Transfer to Subprogram, p.40](#)
[Called Procedures, User's Guide.](#)

Examples In calling program:
0170 **CALL** "SUBR" ,LEN(A\$) ,N ,A\$,T{**ALL**}

 In subprogram "SUBR":
0020 **ENTER** A ,B ,Z\$,N{**ALL**}

ERASE Directive *Delete File/Directory from System*

Format `ERASE name$[,ERR=stmtref]`

Where:

name\$ Name of file or directory to be deleted from the system. String expression. To erase a file from a program library, use [\[LIB\]](#), [p.781](#).

stmtref Program line number or label to transfer control to.



Restrictions: A directory can only be deleted if it does not contain any files. ProvideX is subject to OS rules for the deletion of files or directories. In some operating systems, the **ERASE** directive is not accepted and will not delete the directory.

Description Use the **ERASE** directive to delete the file or directory you name. The disk space that was used by the file or directory is returned to the system. If you erase a file, all data in the file is lost.

WindX supports the use of this directive via the [WDX] tag; e.g., `ERASE "[WDX]somefile.ext" . . .` For more information, see [\[WDX\] Direct Action to Client Machine](#), [p.801](#).



Note: This directive does not apply to any file segments for a multi-segmented file.

See Also [Creating, Deleting, and Renaming Data Files](#), *User's Guide*.

Examples

```
0010 ERASE "PRNTFL"
0030 ERASE "SRTFLL1",ERR=0040
```


ERROR_HANDLER Directive Define Generic Handler

- Formats
1. *Define/Remove Generic Handler*: **ERROR_HANDLER** [*prog\$*[:*entry\$*]]
 2. *Find Current Name*: **ERROR_HANDLER READ** *var\$*

Where:

- entry\$* Optional entry label in the error-trapping program. Define once per session.
- prog\$* Name of the error-trapping program. String expression. Omit the name to cancel the current error handler.
- var\$* String variable. Receives the name of the current error handler.

Description Use the **ERROR_HANDLER** directive to assign a generic error-handling program to be invoked internally by the system whenever an error occurs that is not already handled (i.e., by an **ERR=** statement reference or a **SETERR** directive). If the system is unable to properly load and execute the specified error-handling program, ProvideX will display Error #54: Unable to Load Error Handler.

See Also [START Restart ProvideX, p.328.](#)

Format 1: *Define/Remove Generic Handler*

ERROR_HANDLER [*prog\$*]

This defines an error-handling program to take corrective action and then return to the offending statement via an **EXIT** directive. If you have an error handler program in place when an error occurs without handling instructions, ProvideX calls this program to deal with it.

If you use an **EXIT ERR** directive to return from the error-handling program, the normal error processor is invoked and control can be transferred to Command mode.

To cancel the current error handler, omit the program name (i.e., use **ERROR_HANDLER**). The error handler program remains in effect until a **START** directive is executed. The following is a typical **START_UP** program:

```
0010 PREFIX "===/ MISC/"
0020 IF WHO<>"BOSS" THEN SETESC OFF
0030 ERROR_HANDLER "*ERROR"
0040 ! The asterisk marks "*ERROR" as a Sage Software Canada Ltd. utility
```

Format 2: *Find Current Name*

ERROR_HANDLER READ *var\$*

Use the **READ** format to find out the name of the program currently in effect as the error handler; e.g.,

```
-:ERROR_HANDLER READ A$
-:?a$
*error
```

ESCAPE Directive

Interrupt Program Execution

Format **ESCAPE** [*err_val*]

Where:

err_val Optional numeric value to be placed in the **ERR** variable. (ProvideX interprets this as an error code and the system logic kicks in.)

Description When you use the **ESCAPE** directive in Execution mode, ProvideX suspends execution of the current program, lists the current statement (the line number with the **ESCAPE** directive), and returns you to Command mode. Use the **RUN** directive to have the program resume where it left off.

Simplify the debugging process by placing **ESCAPE** statements strategically in your program. When execution is suspended, you are returned to Command mode, where you can evaluate execution and the values of variables, etc. (up to the line where you placed the **ESCAPE** in your application).

If you use **ESCAPE** in Command mode, ProvideX lists the next statement to be processed, if any. If you specify an error value, the **ESCAPE** directive will generate an error with that specific error value. Use this to provide an error exit in a multi-line function.

The ***ESCAPE** label emulates an **ESCAPE** directive for use as a statement reference.



Note: The '**XT**' system parameter is automatically reset to prevent session termination when the **ESCAPE** directive is used in a program.

See Also

[DEF FN Define Function, p.68](#)
[Labels/Logical Statement References, p.816](#)

Example

```
0100 PRINT "BEGIN"; ESCAPE; PRINT "DONE"
```

When run would yield

```
BEGIN
0100 PRINT "BEGIN"; ESCAPE; PRINT "DONE"
1> Command mode prompt
Entry of a 'RUN' command would yield: DONE
```

EXECUTE Directive

Execute Basic Instruction

Format EXECUTE *statement* \$[,ERR=*stmtref*]

Where:

statement\$ Character string to be processed by the system *as if* entered in Command mode. String expression.

stmtref Program line number or label to transfer control to.

Description Use **EXECUTE** to embed Command mode statements directly into a program. A typical use of this directive is to modify the current program dynamically. When used in a compound statement, **EXECUTE** must be the last directive in the line.



Note: By default ProvideX changes the current program. If the **EXECUTE** directive starts with a line number, ProvideX modifies the current program and the line becomes part of the current program, possibly overwriting the existing code. However, if the system parameter 'EX' is *on*, it modifies the program at level 1 (the main level).



Note: Under WindX, you can use EXECUTE "[WDX] . . ." to encapsulate a directive that is not supported across a WindX connection. See [\[WDX\] Direct Action to Client Machine, p.801](#).

Examples

```
0010 LET X$=LST(PGM(TCB(4)+1));EXECUTE X$(5),ERR=*END
0100 IF WDX THEN EXECUTE "[WDX]SET_PARAM 'SD' "
```

EXIT Directive *Terminate Subprogram and Return*

Format EXIT [ERR | *err_val*]

Where:

err_val Numeric expression whose value will be returned as an error status to the program initiating the subprogram (e.g., via a **CALL** directive).



Restriction: You can only use this directive in *subprograms*. Otherwise, ProvideX returns Error #37: Directive can only execute in subprogram.

Description Use the **EXIT** directive in a subprogram to terminate the subprogram and return control to the initiating program.

You can have the subprogram return an error code value to the calling program by using **EXIT ERR** or by specifying the error value following the **EXIT** directive. Use an integer from 0 to 32767 for the error value. To have the calling program process an error value other than zero, use the **ERR=** option in the **CALL** directive (or use **SETERR numeric expression**). If the value is 0 *zero*, ProvideX does not retry processing.

When you use **EXIT** in a compound statement, it must be the *final* directive:

```
9000 PRINT "Subroutine done"; EXIT
```

See Also [CALL Transfer to Subprogram, p.40](#)
[PERFORM Call Subprogram, Pass Variables, p.243](#)
[Called Procedures, User's Guide.](#)

Examples 10000 ! In subprogram "SUBPR"
10005 TEST_EXIT:
10010 SETERR UNKNOWN_ERROR
10020 ENTER TEST\$
10030 LET T=NUM(TEST\$(5,6))
10040 EXIT
10050 UNKNOWN_ERROR:
10060 EXIT ERR

EXITTO Directive *End Loop, Transfer Control*

Format EXITTO [*stmtref*]

Where:

stmtref Program line number or label to transfer control to.

Description The **EXITTO** directive terminates the currently active **FOR..NEXT**, **GOSUB..RETURN**, **REPEAT..UNTIL** or **WHILE..WEND** loop prematurely and transfers control to the statement number indicated.

EXITTO lets you terminate one of these processes early by removing its associated entry from the top of the stack. If there is no active entry on the stack, ProvideX returns Error #27: Unexpected or incorrect WEND, RETURN, or NEXT.

When used in a compound statement, **EXITTO** must be the final directive.

See Also [FOR..NEXT Loop While Incrementing, p.134](#)
[GOSUB.. Execute Subroutine, p.141](#)
[REPEAT..UNTIL Repetitive Execution, p.287](#)
[WHILE..WEND Repeat Statements, p.375](#)
Flow Overrides, User's Guide.

Example

```
00010 BEGIN
00020 FOR i=1 TO 10
00030 INPUT x
00040 IF CTL=4 \
      THEN EXITTO 0060
00050 acc+=x
00060 NEXT i
00070 IF i>1 \
      THEN avg=acc/(i-1)
00080 PRINT avg
```

EXTRACT Directive

Read and Lock Data

Format **EXTRACT** (*chan*[,*fileopt*])*varlist*

Where:

chan Channel or logical file number of the file from which to read the data.

fileopt Supported file options (see also, **File Options**, p.810):
BSY=stmref Traps Error #0: Record/file busy
DOM=stmref Missing record transfer
END=stmref End-Of-File transfer
ERR=stmref Error transfer
IND=num Record index
KEY=string\$ Record key
KNO=num | name\$ File access key number (*num*) or name (*name\$*)
REC=name\$ Record prefix (**REC=VIS(string\$)** can also be used)
RNO=num Record number
RTY=num Number of retries (one second intervals)
SIZ=num Number characters to read
TBL=stmref Data translation table
TIM=num Maximum time-out value in integer seconds.

varlist Comma-separated list of variables, literals, and **IOL=** options.

Description Use **EXTRACT** to read data from the file you specify as the channel. When ProvideX reads the data, it is split into one or more fields (either separated by the current delimiter or in an embedded IOList format) with the contents of the first field placed into variable 1, the second field into variable 2, and so on.

ProvideX automatically converts numeric data when moving it into numeric variables while processing the **EXTRACT** directive. Numeric data converted during an **EXTRACT** directive does not use the '**DP**' **Decimal Point Symbol** or '**TH**' **Thousands Separator** system parameters for European decimal settings.

If you want a field to be skipped, use an *asterisk* * as a place holder for a variable name.

If you specify more variables than there are fields in the record, ProvideX will initialize the additional variables to either zero (if a numeric variable) or a null string (if a string variable). The **EXTRACT** directive will access the record where the file pointer is pointing, or it will take the record specified using the **KEY=** or **IND=** options. The file pointer remains on the extracted record after it is read.

Use the **KNO=** option to change the current file access key.



Note: **EXTRACT** locks the record being read to prevent other users from using a **FIND**, **FIND RECORD**, **READ**, **READ RECORD**, **EXTRACT RECORD** or another **EXTRACT** to access it. This lock stays active until the next I/O request for the same file or until the file is closed. Using a **KEY=** option or **READ**, **FIND** or **EXTRACT** statement to retrieve the next record while a record is locked will result in the locked record being returned instead. You can enable read access for records that have been extracted by setting the 'XI' parameter.

See Also

[FIND Locate and Read Data, p.131](#),
[READ Read Data from File, p.271](#),
[EXTRACT RECORD Read-Lock Data Record, p.128](#),
[READ RECORD Read Record from File, p.275](#)
[KEY\(\) Function, p.470](#).

EXTRACT RECORD Directive Read-Lock Data Record

Format **EXTRACT RECORD** (*chan*[,*fileopt*])*var*\$

Where:

chan Channel or logical file number of the file from which to read the data.

fileopt Supported file options (see also, [File Options, p.810](#)):
DOM=stmtrf Missing record transfer
END=stmtrf End-Of-File transfer
ERR=stmtrf Error transfer
IND=num Record index
KEY=string\$ Record key
KNO=num | name\$ File access key number (*num*) or name (*name\$*)
REC=name\$ Record prefix (**REC=VIS(string\$)** can also be used)
RNO=num Record number
RTY=num Number of retries (one second intervals)
SIZ=num Number characters to read
TBL=stmtrf Data translation table
TIM=num Maximum time-out value in integer seconds.

var\$ String variable. Receives the contents of the record being read.

Description Use the **EXTRACT RECORD** directive to read a record from the file you specify (channel). ProvideX will return the record's complete data portion in the string variable you specify.

Apply the **EXTRACT RECORD** statement when dealing with native-mode operating system files, when exchanging data with non-ProvideX applications, or when you want to read a complete record (including data field separators). This directive will access the record where the file pointer is pointing, or it will take the record specified using the **KEY=** or **IND=** options. The file pointer remains on the extracted record after it is read.

Use the **KNO=** option to change the current file access key.

Note: This directive *locks* the record being read to prevent other users from using a **FIND**, **FIND RECORD**, **READ**, **READ RECORD**, **EXTRACT RECORD** or another **EXTRACT RECORD** to access it. This lock remains active until the next I/O request for the same file or until the file is closed. Using a **KEY=** option or **READ**, **FIND** or **EXTRACT** statement to retrieve the next record while a record is locked will result in the locked record being returned instead. You can enable read access for records that have been extracted by setting the '**XI**' parameter.



See Also [FIND Locate and Read Data, p.131](#),
[READ Read Data from File, p.271](#),
[EXTRACT Read and Lock Data, p.126](#)
[KEY\(\) Function, p.470](#).

Examples

```
0010 OPEN (1) "OLDFIL"  
0020 OPEN (2) "NEWFIL"  
0030 LOCK (2)  
0040 EXTRACT RECORD (1,END=1000) R$  
0050 WRITE RECORD (2) R$  
0060 GOTO 0040  
1000 CLOSE  
1010 END
```



FILE Directive

Create New File from File Descriptor

Format **FILE** *file_info* [,ERR=*stmtref*]

Where:

file_info Contents provide the internal file description for the file to create.
String expression.

stmtref Program line number or label to transfer control to.

Description Use the **FILE** directive to define a file based on the contents returned by the **FID()** or **FIB()** functions. These functions return character strings with the file type, size, and format.



Warning: The **FILE** directive will *not* recreate an embedded data dictionary. When you have an embedded data dictionary, use a **PURGE** or **REFILE** directive instead, to clear the data from an existing file and preserve the dictionary.



Note: The format of the **FID()** value will depend on the current emulation mode you are using for ProvideX. To avoid potential problems when running in emulation modes, use the value returned from the **FIB()** function instead of the **FID()** function.



Note: This directive is not supported by WindX. When you need to create files on a WindX PC, encapsulate the command in an **EXECUTE "[WDX] . . ."** directive.

See Also

[FIB\(\) Function, p.434](#),
[FID\(\) Function, p.438](#),
[PURGE Clear Data from a File, p.263](#),
[REFILE Clear Data from File, p.278](#)
[\[WDX\] Tag, p.801](#)
[Creating, Deleting, and Renaming Data Files, User's Guide](#)

Examples

```
0010 OPEN (2) "PRNTFL"
0020 F$=FID(2)
0030 CLOSE (2)
0040 ERASE "PRNTFL"
0050 FILE F$
0060 OPEN (2)F$(25,60)
```

FIND Directive

Locate and Read Data

Format `FIND (chan[,fileopt])varlist`

Where:

chan Channel or logical file number of the file from which to read the data.

fileopt Supported file options (see also, [File Options, p.810](#)):

- BSY=***stmtref* Traps Error #0: Record/file busy
- DOM=***stmtref* Missing record transfer
- END=***stmtref* END-OF-FILE transfer
- ERR=***stmtref* Error transfer
- IND=***num* Record index
- KEY=***string\$* Record key
- KNO=***num* | *name\$* File access key number (*num*) or name (*name\$*)
- REC=***name\$* Record prefix (**REC=VIS(*string\$*)** can also be used)
- RNO=***num* Record number
- RTY=***num* Number of retries (one second intervals)
- SIZ=***num* Number of characters to read
- TBL=***stmtref* Data translation table
- TIM=***num* Maximum time-out value in integer seconds.

varlist Comma-separated list of variables, literals, and **IOL=** options.

stmtref Program line number or label to transfer control to.

Description Use **FIND** to read data from the file (channel) you specify. When ProvideX reads the data, it's split into one or more fields (either separated by the current delimiter or defined by an embedded format with headers, etc.). The contents of the first field are placed in variable 1, the second field in variable 2, and so on.

ProvideX automatically converts numeric data when executing a **FIND** statement and moving numerics into variables. Numeric data converted during a **FIND** directive does not use the **'DP' Decimal Point Symbol** or **'TH' Thousands Separator** system parameters for European decimal settings.

If you want to skip a field, use an *asterisk* * as a place holder for the variable name. If you include more variables in a **FIND** directive than there are fields in the record, ProvideX initializes the additional variables to either zero (for a numeric variable) or a null string (for a string variable). If successful, **FIND** advances the file position to the next record (or the record you specify if you use a **KEY=** or **IND=** option). Use the **KNO=** option to change the current file access key.



Note: If the record is not found when reading using a **KEY=** or **IND=** option, the current file position is *not changed* (unlike **READ**).

See Also [EXTRACT Read and Lock Data, p.126](#)
[READ Read Data from File, p.271](#).

Examples 0410 FIND (1,ERR=1000,DOM=1200)A,B,*,*,E\$

FIND RECORD Directive *Locate & Read Data Record*

Format **FIND RECORD** (*chan* [, *fileopt*]) *var*\$

Where:

chan Channel or logical file number of the file from which to read the data.

fileopt Supported file options (see also, [File Options, p.810](#)):
DOM=stmtref Missing record transfer
END=stmtref End-of-File transfer
ERR=stmtref Error transfer
IND=num Record index
KEY=string\$ Record key
KNO=num | name\$ File access key number (*num*) or name (*name\$*)
RNO=num Record number
RTY=num Number of retries (one second intervals)
SIZ=num Number of characters to read
TBL=stmtref Data translation table
TIM=num Maximum time-out value in integer seconds.

var String variable. Receives the contents of the record being read.

Description Use the **FIND RECORD** directive to read a record from the file (channel) you specify. The record's complete data portion will be returned in the string variable you name.

Apply the **FIND RECORD** statement when dealing with native-mode operating system files, when exchanging data with applications other than ProvideX, or when you want to read a complete record including data-field separators.

If successful, **FIND RECORD** advances the file position to the next record (or the record you specify in a **KEY=** or **IND=** option). Use the **KNO=** option to change the current file access key.



Note: If the record is not found when reading using a **KEY=** or **IND=** option, the current file position is *not changed* (unlike **READ**).

See Also [EXTRACT Read and Lock Data, p.126](#)
[READ Read Data from File, p.271](#).

Examples

```
0020 OPEN LOCK (2) "NEWFIL"
0030 FIND RECORD (1, END=1000) R$
0040 WRITE RECORD (2) R$
0050 GOTO 0030
1000 CLOSE (1), (2)
1010 END
```

FLOATING POINT Directive *Switch to Scientific Notation*

Format **FLOATING POINT**

Description Use the **FLOATING POINT** directive to have ProvideX start using scientific notation with all numeric data. In scientific notation, a number is represented as a fraction times ten to some power (e.g., $.nnnnn \cdot 10^{xx}$ is displayed as $.nnnnnE_{xx}$).

In **FLOATING POINT** mode, no rounding is done on numeric calculations. If the numeric output is unformatted, ProvideX uses scientific notation (e.g., $.314159E+01$).

Use the **PRECISION** directive to end **FLOATING POINT** mode.

See Also [BEGIN Reset Files and Variables, p.32](#),
[CLEAR Reset Variables, p.54](#),
[PRC System Variable, p.569](#)
[PRC\(\) Function, p.503](#)
[PRECISION Change Current Precision, p.248](#),
[RESET Reset Program State, p.288](#).



Note: **FLOATING POINT** notation will change to standard decimal notation during a **RESET** operation.

Examples

```
0030 FLOATING POINT
0040 A=5/3
0050 PRINT A
RUN
      .166666666666667E+01
0010 PRECISION 2 ! Sets precision to two
0020 ROUND ON ! Sets rounding default mode
0030 A=5/3
0040 FLOATING POINT
0050 PRINT A
-:RUN
      .167E+01
```

FOR..NEXT Directive

Loop While Incrementing

Format

1. *Conventional Syntax*: **FOR** *var*=*first* **TO** *last* [**STEP** *val*] **..NEXT** [*var*]

2. *Simplified Syntax*: **FOR** *var* **..NEXT** [*var*]

Where:

first Initial value of the variable *var*. Numeric expression.

last Ending value of the variable *var* (value that ends the loop when achieved).
Numeric expression.

NEXT Directive to end the loop. Optional *var* must match current **FOR** *var*.

TO Keyword required for *Format 1*, not case-sensitive.

STEP Optional keyword, not case-sensitive. Sets specific increment/decrement value (default is 1).

val Optional value by which the variable will be incremented/decremented with each pass through the loop. The default is 1.

var Numeric control variable to be incremented/decremented with each pass through the loop.

Description

Use the **FOR** directive (either *conventional* or *simplified* iteration format) to define the start of a repetitive loop of instructions in a program. If you specify a variable with a **NEXT** directive, it must match the variable in the **FOR** directive. The **NEXT** directive can appear anywhere in the program *except* where it would be executed inside another **FOR/NEXT** loop, a **GOSUB/RETURN** routine, a **WHILE/WEND** loop, or a **REPEAT/UNTIL** loop. The control variable can be omitted from the **NEXT** directive because the increment/decrement of the **FOR** *var* is assumed automatically. However, the **NEXT** *var* is useful for readability purposes, especially if it appears within a *nested* loop structure.

Use the **EXITTO** directive to halt a **FOR/NEXT** loop without performing all iterations. When ProvideX executes an **EXITTO** directive, it removes the top entry from the **FOR/NEXT** stack and ends the current **FOR/NEXT** loop.

Format 1: For Loop, Conventional Syntax

FOR *var*=*first* **TO** *last* [**STEP** *val*] **..NEXT** [*var*]

When using conventional syntax, the keyword **TO** is required in order to define *first* and *last* values in the loop.

The **STEP** value, if specified, is evaluated and saved as the increment (or *decrement if negative*). A default **STEP** value of 1 *one* is used when the increment/decrement is not declared. An increment of 0 *zero* is invalid and results in an Error #44: Invalid step value.

After the **FOR** directive is executed, the current program statement position is saved and execution continues. When ProvideX encounters **NEXT** with the same variable as in the **FOR** directive (or no variable name) it increments/decrements the value. If the new value does not exceed the end value (or fall below it, if decremented), control transfers back to the **FOR** directive to continue the loop. Otherwise the **FOR/NEXT** loop ends.

Examples:

```
0010 FOR I = 1 TO 10
0020 PRINT I,
0030 NEXT I
yields: 1 2 3 4 5 6 7 8 9 10
```

The following example strips trailing spaces from the A\$:

```
0030 A$=.....
0040 GOSUB 1000
.....
1000 IF A$="" THEN I=0; RETURN
1010 FOR I=LEN(A$) TO 1 STEP -1
1020 IF A$(I,1)<>" " THEN EXITTO 1040
1030 NEXT
1040 RETURN
```

This uses nested **FOR/NEXT** loops to generate the string **B\$** from **A\$** reversed:

```
0010 INPUT "Enter character string:",A$
0020 IF A$="" THEN STOP
0030 B$=A$
0040 FOR I=1 TO LEN(A$)
0050 FOR J=LEN(A$) TO 1 STEP -1
0060 B$(I,1)=A$(J,1)
0070 NEXT J; NEXT I
0080 PRINT "The answer is: ", B$
0090 GOTO 0010
```



Note: The test to determine if the loop will be terminated occurs when the **NEXT** directive is encountered; therefore, if the initial value of the loop control variable exceeds the ending value, the loop will execute once.

Format 2: *For Loop, Simplified Syntax*

FOR *var* ..NEXT [*var*]

This format executes the logic immediately following the **FOR** by the number of times specified in the numeric value *var*; therefore, **FOR 1** will execute the loop once, and **FOR 5** will execute the loop 5 times. A value of zero causes the loop to be skipped. An error is generated if the numeric value is not an integer or it is less than zero.

If *var* is a simple numeric variable, the system will first set *var* to 1, then increment up to its initial value. When *var* = 5, the loop will execute 5 times, with *var* starting at 1 and incrementing by 1 through each iteration to 5. At the completion of the loop, *var* will equal its initial value. Regardless of whether a simple numeric variable is used, **TCB(19)** will contain the current iteration count during the loop.

Example 1:

```
N = LEN(X$)
FOR N
  IF X$(N,1) = "X" THEN X$(N,1) = "Y"
NEXT
```

Example 2:

```
TOT = 0
FOR DIM(READ MAX(Balance))
  TOT += Balance[TCB(19)]
NEXT
```

Should a loop using a defined variable be prematurely exited (via **BREAK**, **POP**, or **EXITTO**), the variable will remain at its last value; e.g.,

```
N = 10
FOR N
  IF X$[N] = "" BREAK
  ...
NEXT
```

When the **IF** condition is satisfied and the **FOR** loop is exited via the **BREAK**, the value in *N* will be the index into the array.

See Also

[BREAK Immediate Exit of Loop, p.33](#),
[CONTINUE Initiates Next Iteration of Loop, p.57](#)
[EXITTO End Loop, Transfer Control, p.125](#)
[Loop Structures, User's Guide.](#)

FUNCTION Directive

Declare Object Method

Formats

1. *Declare Method*: **FUNCTION** *method(args) logic*
2. *Method with PERFORM Behaviour*: **FUNCTION PERFORM** *method(args) logic*
3. *Local Method*: **FUNCTION LOCAL** *method(args) logic*
4. *COM Event Method*: **FUNCTION** *method(args) logic FOR EVENT* {*event\$*|**SAME**}
5. *End Method Declaration*: **FUNCTION END**

Where:

(args)	Optional list of arguments to be used in the logic when the method is actually invoked. Parentheses are required with/without arguments.
END	Keyword indicating the end of a method declaration.
event\$	Name of the corresponding COM event.
FOR EVENT	Keywords indicating that method is associated with given COM event.
LOCAL	Keyword indicating that method is only to be called within the object.
logic	Procedure associated with method. Method should return a value; if not, the system forces 0 or "".
method	Name of method that the object can perform. (In <i>Object Oriented Programming</i> , functions are referred to as <i>methods</i> .)
PERFORM	Keyword indicating that logic is to be loaded/executed as in a PERFORM .
SAME	Keyword to use if the <i>method</i> name matches <i>event\$</i> name.

Description

The **FUNCTION** directive is used to declare a method for an object in *Object Oriented Programming* (OOP). Associated logic is to be called when the method is invoked; e.g.,

```
FUNCTION Find(X$) LookupCust
... ..
LookupCust:
ENTER Cst_id$
... .. ! Logic to find the client
RETURN sts ! Return value indicates success
```

Alternatively, the function logic can directly follow the **FUNCTION** declaration; e.g.,

```
FUNCTION Find(X$)
ENTER Cst_id$
... .. ! Logic to find the client
RETURN sts ! Return value indicates success.
```

The declaration ends with a **FUNCTION END** directive for the method itself, by the start of the next method declaration, or when **END DEF** is reached at the end of the object definition; e.g.,

```
DEF CLASS "MyObj"
FUNCTION MyFirstMethod()
a=b, d=f
```

```

if ... .. etc
FUNCTION MySecondMethod()
a=b, d=f
if ... .. etc
FUNCTION END
END DEF

```

Every method should return a value. The value can take the form of a string or numeric value depending on the name associated with the function (string functions must end with \$). If no **RETURN** value is specified, then the system will return a value of zero for numeric functions and "" (null) for string functions.



Note: As a general rule of thumb, methods should return a non-zero value when successfully executed. This allows for logic such as:

```
IF NOT(Cst'Find("ABCD")) THEN GOTO Bad_cust.
```

When arguments are used in the definition, then the type/number should normally match variables in the application code. Multiple definitions of the same method name can be specified, as long as each method has a different parameter list. In order to determine which method to actually use, ProvideX attempts to match up the parameter lists specified with the variables provided in the application.

If an *asterisk* * is used in the definition in place of the argument list (e.g., FUNCTION readbykey(*)), the method will be invoked regardless of the type/number of variables to be matched in the corresponding logic.

FUNCTION PERFORM indicates that the function logic is to be loaded and executed (as in a **PERFORM** directive). All variables will be shared with the calling program.



Note: The **PERFORM** format violates the general rules of OOP *encapsulation*.

FUNCTION LOCAL indicates that the function is only to be called internally from within the object. It cannot be called externally.

Examples

```

FUNCTION Find(X$) LookupByName
FUNCTION Find(X) LookupByNumber
... ..
LookupByName:
ENTER Cst_id$
... .. ! Logic to find the client by name
RETURN ...
LookupByNumber:
ENTER Cst_id
... .. ! Logic to find the client by number
RETURN ...

```

See Also

DEF CLASS Define Object Class, *p.65*
ON EVENT Event Processing, *p.228*
Data Integration, *User's Guide*.

GET_FILE_BOX Directive

Ask for Filename

Formats

1. *Create File Box*: `GET_FILE_BOX path$,dir$,window$[ex_list$[,def_ex$]][,ERR=stmtref]`
2. *Check File Box*: `GET_FILE_BOX READ path$,dir$,window$[ex_list$[,def_ex$]][,ERR=stmtref]`
3. *Write to File Box*: `GET_FILE_BOX WRITE path$,dir$,window$[ex_list$[,def_ex$]][,ERR=stmtref]`
4. *Multiple Selection*: `GET_FILE_BOX READ LIST path$,dir$,window$[ex_list$[,def_ex$]][,ERR=stmtref]`
5. *Directories/Folders Only*: `GET_FILE_BOX DIRECTORY path$,dir$,prompt$[,root$]][,ERR=stmtref]`

Where:

<i>dir\$</i>	Initial directory to display. String expression.
<i>def_ext\$</i>	Default file extension to apply when creating a file. Optional. String expression.
<i>ex_list\$</i>	List of file extensions. Optional. String expression. Comma (or any character) as delimiter.
<i>path\$</i>	String variable that contains the file path. Initialize this prior to executing this directive. See the note, below.
<i>prompt\$</i>	Text appearing above the directory display.
<i>root\$</i>	<i>For Windows XP or later.</i> Optional highest level directory in which browsing can occur. (This parameter overrides <i>dir\$</i> .)
<i>stmtref</i>	Program line number or label to transfer control to.
<i>window\$</i>	Window title. String expression.



Note: `GET_FILE_BOX` is supported when you use WindX with UNIX file systems and supported calls to [WDX]. See [\[WDX\] Direct Action to Client Machine, p.801](#).

Description

Use the `GET_FILE_BOX` directive to display a standardized window for the user, allowing the entry or selection of a file or directory in the system.

Format 1: Create

`GET_FILE_BOX path$,dir$,window$[ex_list$[,def_ex$]][,ERR=stmtref]`



Note: The *path\$* variable will receive the full pathname of the file selected. Because its initial contents will be used as the *default* pathname, initialize it prior to executing this directive.

Use this format to select a file. If the initial directory string is null, ProvideX uses the current directory. Use the form `Description|*.XXX`, to list standard file extensions. You can include multiple extensions, comma-delimited (with the last extension terminated by a comma). If desired, choose a character other than a comma to delimit each entry. ProvideX uses the last character in *ex_list\$* (the comma or your choice) as the delimiter.

Format 2: *Check*

GET_FILE_BOX READ *path\$,dir\$,window\$[ex_list\$[,def_ex\$,,][,ERR=stmtref]*

Use the **READ** format to make sure that the file you want returned exists; e.g.,

```
0010 GET_FILE_BOX READ X$, "C:\Program Files\Sage
      Software\ProvideX\","File to View"
```

Format 3: *Write*

GET_FILE_BOX WRITE *path\$,dir\$,window\$[ex_list\$[,df_ex\$,,][,ERR=stmtref]*

Use the **WRITE** format to save or rewrite the file. If you do this and a user selects a file that already exists, ProvideX will confirm that the file is to be overwritten prior returning to the program; e.g.,

```
0010 GET_FILE_BOX WRITE X$,LWD,"Report file",
      "Report files|.RPT,All files|.*.","RPT"
```

This allows the user to have a filename with the default extension `.RPT` and gives the user a list of files of two types—Report files (`*.RPT`) and All files (`*.*`).

Format 4: *Multiple Selection*

GET_FILE_BOX READ LIST *path\$,dir\$,window\$[ex_list\$[,df_ex\$,,][,ERR=stmtref]*

Use **LIST** to specify multiple selections for **GET_FILE_BOX READ**. If a single entry is selected, then *path\$* will contain the full path of the file (as with a **GET_FILE_BOX READ**). If more than one entry is selected, then *path\$* will consist of the full path of the first item terminated by **SEP**, followed by a comma-delimited list of the other selected files.



Note: Windows ignores selected directories. This format is not supported in the text version of **GET_FILE_BOX**.

Format 5: *Directories/Folders Only*

GET_FILE_BOX DIRECTORY *path\$,dir\$,prompt\$[,root\$][,ERR=stmtref]*

Use this format to select a directory/folder from a list. The contents of *prompt\$* is displayed inside the dialog box above the directory display. The dialog box caption reads: "Browse for folder".

For Windows XP or later. The optional *root\$* parameter sets the highest level directory in which browsing can occur (overriding what is set in *dir\$*). The terms "My Computer" and "My Documents" can be used as well as the full path to an existing folder.

GOSUB Directive

Execute Subroutine

Format **GOSUB** *stmtref*

Where:

stmtref Program line number or label to transfer control to.

Description Use **GOSUB** to access a subroutine embedded in your current program. The current location in the program is saved on a stack and control is passed to the statement number or label you specify (or if no statement exists with the number specified, to the next higher statement). ProvideX continues execution from there until a **RETURN** is executed. Then it returns control to the position saved on the stack.

If you want to exit from a **GOSUB** subroutine without returning to the calling point, you can use the **EXITTO** directive to remove the return point from the stack. You must use either a **RETURN** or an **EXITTO** directive to terminate each **GOSUB** subroutine.

Do not place a **RETURN** directive inside a **FOR..NEXT** loop or a **WHILE..WEND** loop. There is no limit (apart from memory space) to the number of **GOSUB..RETURN** entries that you can have on the stack.



Note: The **BEGIN**, **CLEAR**, **RESET**, **STOP**, and **END** directives reset all entries in the **GOSUB..RETURN** stack.

See Also **RETURN Subroutine/Function Return, p.291**
Called Procedures, User's Guide

Examples

```
D$=" ",D1$=" ",D2$=" "
-:/
0100 INPUT "Enter starting date DD/MM/YY: ",D$
0110 GOSUB 1000
0120 LET D1$=D$
0130 INPUT "Enter ending date DD/MM/YY: ",D$
0140 GOSUB 1000
0150 LET D2$=D$
0160 PRINT "DONE"; STOP
1000 REM Subroutine
1010 IF LEN(D$)<>8 THEN GOTO 1090
1020 IF D$(3,1)<>"/" OR D$(6,1)<>"/" THEN GOTO 1090
1030 LET D=NUM(D$(1,2),ERR=1090),M=NUM(D$(4,2),ERR=1090),Y=NUM(D$(7,2),ERR=
1030:1090)
1040 IF D<1 OR D>31 THEN GOTO 1090
1050 IF M<1 OR M>12 THEN GOTO 1090
1060 RETURN
1090 PRINT "Invalid. Restart.."
1100 EXITTO 0100
-:RUN
Enter starting date DD/MM/YY: 05/03/99
Enter ending date DD/MM/YY: 31/03/99
-:DONE
```

GOTO Directive

Transfer within Program

Format **GOTO** *stmtref*

Where:

stmtref Program line number or label to transfer control to.

Description Use the **GOTO** directive to have ProvideX transfer execution to the given statement number or label. If the line doesn't exist, then the statement with the next higher number will be used.

The **GOTO** directive can be used in either Execution or Command mode. In Command mode, the **GOTO** directive causes execution to begin at the statement number specified upon execution of the next **RUN** directive.

If you use **GOTO** in a compound statement, it must be the final directive.

Examples

In Execution mode:

```
0010 INPUT "Enter a number: ",A
0020 IF A=0 THEN PRINT "DONE"; STOP
0030 PRINT A," multiplied by 2 is ",A*2
0040 GOTO 0010
```

In Command mode:

```
 -:GOTO 10
 -:RUN
Enter a number: 1
1 multiplied by 2 is 2
Enter a number: 3
3 multiplied by 2 is 6
Enter a number: 0
DONE
 -:
```

GRID Directive

Control Grid

Formats

1. *Define/Create*: **GRID** *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]
2. *Remove*: **GRID REMOVE** *ctl_id*[,ERR=stmtref]
3. *Disable/Enable*: **GRID** {**DISABLE** | **ENABLE**} *ctl_id*[,ERR=stmtref]
4. *Lock/Unlock*: **GRID** {**LOCK** | **UNLOCK**} *ctl_id*,col,row[,width,height][,ERR=stmtref]
5. *Hide/Show*: **GRID** {**HIDE** | **SHOW**} *ctl_id*[,ERR=stmtref]
6. *Force Focus*: **GRID GOTO** *ctl_id*[,col,row][,ERR=stmtref]
7. *Signal on Focus*: **GRID SET_FOCUS** *ctl_id*, *ctl_val*[,ERR=stmtref]
8. *Add Row/Column*: **GRID ADD** *ctl_id*,col,row[,ERR=stmtref]
9. *Load*: **GRID LOAD** *ctl_id*,col,row,contents\$,[ERR=stmtref]
10. *Delete Row/Column*: **GRID DELETE** *ctl_id*,col,row[,ERR=stmtref]
11. *Clear*: **GRID CLEAR** *ctl_id*,col,row[,width,height][,ERR=stmtref]
12. *Read First Cell*: **GRID READ** *ctl_id*,col,row,var\$,eom\$[,ERR=stmtref]
13. *Read Next*: **GRID READ NEXT** *ctl_id*,col,row,var\$,eom\$[,ERR=stmtref]
14. *Select Range*: **GRID SELECT** *ctl_id*,col,row[,width,height][,ERR=stmtref]
15. *Select Read*: **GRID SELECT READ** *ctl_id*,col,row[,ERR=stmtref]
16. *Select Read Next*: **GRID SELECT READ NEXT** *ctl_id*,col,row[,ERR=stmtref]
17. *Select Reset*: **GRID SELECT RESET** *ctl_id*[,ERR=stmtref]
18. *Find or Retrieve*: **GRID FIND** *ctl_id*,col,row,var\$[,ERR=stmtref]
19. *Write Setting*: **GRID WRITE** *ctl_id*,col,row,contents\$[,ERR=stmtref]
20. *Report all Changes*: **GRID AUTO** *ctl_id*[,ERR=stmtref]

Where:

- @(col,ln,wth,ht)** Position and size of the grid region on the screen. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.
- col,row** Column and row refer to cell location / coordinates. Numeric expressions.
- contents\$** Value(s) to be written to the cell(s) or settings. String expression(s).
- ctl_id** Unique logical identifier for the grid (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the **F4** key) or **Negative CTL Definitions, p.817**. Use this value with the apostrophe operator to access various **Grid Properties**.
- ctl_val** Alternate CTL value.
- ctrlopt** Control options. Supported options for **GRID** include:
ERR=stmtref Error transfer
FMT=def\$ (See **Format Definition, p.145**)

- FNT**=*"font,size[,attr]"* Font name, size, optional properties. Refer to the '**FONT**' Mnemonic, [p.609](#) for details.
- MNU**=*ctl* CTL value associated with right-click menu event.
- OPT**=*char\$* (See [GRID OPT= Settings](#))
- SEP**=*char\$* Delimiter character. Hex or ASCII string value; e.g.,
 GRID 7050 ,@(5,5,35,15) ,SEP=":" . . .
 GRID 7050 ,@(5,5,35,15) ,SEP=\$3A\$. . .
 GRID 7050 ,@(5,5,35,15) ,SEP=MY_SEP\$. . .
- The default delimiter is the **SEP** character (e.g., \$8A\$).
- eom\$* **EOM** (End-of-Message) character sequence. Hex string expression (e.g., \$0D\$ for the **Enter** key).
- stmtref* Program line number or label to transfer control to.
- var\$* String variable. Receives cell values returned for **FIND** and **READ**.

GRID OPT= Settings

Available attribute/behaviour settings are listed below. Some characters may be combined. Invalid settings are ignored.

- "A" Auto signal is *on*. ProvideX returns a signal on changes.
- "B" Grid has no border or frame.
- "D" *Disabled*. Grid is not accessible to the user.
- "G" *Global*. Keep active when focus changes to a new/non-concurrent window.
- "H" *Hide*. Grid is not displayed but is accessible programmatically.
- "s" *Scroll*. Grid can scroll within a resizable/scrollable dialogue box.
- "T" Strip trailing spaces. (*Supported only by GRID WRITE*).
- "X" *Signal on exit*. Signal when focus exits from the grid.
- "Z" Cursor changes to "resize" pointer if within 4 pixels from the edge of the control.

Description

Use the **GRID** directive to create a table of cells in columns and rows; i.e., a spreadsheet input format. (For simpler tables use a [LIST_BOX Report View, p.189](#).) The **GRID** control object is a two-dimensional array of *multi_line* input fields (default), *check_boxes*, *buttons*, *drop_boxes* or any combination of these control objects. Each cell can be of a different type. The **GRID** treats as a null value that has a length of 0 or consists of zero "0", comma ",", or dot "." characters. For some **GRID** syntax (e.g., **GOTO** and **FIND**), *both* column and row coordinates are required (to act as pointers to a particular cell). For other syntax, one or both of the values can be zero "0". The following chart shows how ProvideX interprets the column and row values.

Column, Row Values	How ProvideX interprets these ...
<i>col=0, row=0</i>	All columns and rows respectively; i.e., GRID DELETE 100,0,0 deletes the entire grid whose control ID is 100.
<i>col=1, row=1</i>	The topmost cell is 1,1

Column, Row Values	How ProvideX interprets these ...
<i>col</i> and/or <i>row</i> = -1	By default, there are also column and row headers you can gain access to using column and/or row = -1. You cannot include these headers in any range specifications.
<i>col</i> = <i>n</i> , <i>row</i> =0 (<i>col</i> <> 0, <i>row</i> = 0 or omitted)	Entire column; i.e., GRID DELETE 100, 3, 0 deletes column 3 of the grid whose control ID is 100. GOTO and FIND formats of the grid directive require both column and line coordinates. You cannot omit the row in those formats.
<i>col</i> =0, <i>row</i> = <i>n</i> (<i>col</i> = 0 or omitted, <i>row</i> <> 0)	Entire row; i.e., GRID DELETE 100, 0, 2 deletes row 2 of the grid whose control ID is 100. GOTO and FIND formats of the grid directive require both column and line coordinates. You cannot omit the column in those formats.
<i>col</i> = <i>n</i> , <i>row</i> = <i>n</i> (<i>col</i> and <i>row</i> are <> 0)	Column <i>and</i> row; i.e., GRID DELETE 100, 3, 2 deletes <i>both</i> an entire column (column 3) and an entire row (row 2) of the grid whose control ID is 100. GRID GOTO 100, 3, 2 sets focus on the cell located in column 3, row 2.

When setting a grid property, setting both the row and column to zero allows you to set that property as a default for the entire grid (or for a specified column if the row is set to zero and column is set to a non-zero number). If a specific row (*row*=*n*, *column*=0) or an individual cell (*row*=*n*, *column*=*n*) is specified, then defaults no longer apply to these cells (rows cannot have default settings, but you can set values for all cells in the row), so if a new property default is set for a column or the entire grid, then the row or individual cells will be excluded.



Note: Each cell in a grid represents a minimum of 140 bytes of information. In a client-server, this volume can be very costly in a low-bandwidth situation.

Format Definition

The **FMT=def\$** option allows you to define the format for cells in the grid. In the following syntax, the round and square brackets are part of the format:

FMT=[*col_title*](*cell_type*:*col_name* \$) *Alignment Width*

Where:

Alignment Alignment character: **L** (left), **C** (centred), **R** (right). **Default:** **L**.

cell_type Cell type for the column. The default cell type is "Normal". See *Cell Types* for a list of available cell types.

col_title Text string appearing as the title in the top row of the grid.

col_name \$ Column name. This can be a string or numeric variable.

width Width of the column in .

Example:

```
"[Client ID](Multi_line:CST_ID$)L10 [Name](Normal:CST_NAME$)L10"
```

Grid Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. Most cell attributes are only accessible through the implementation of properties. For example, drag and drop functionality requires the **'DraggedColumn**, **'DraggedRow**, **'DroppedOnColumn**, and **'DroppedOnRow**.



Note: In order to access many grid properties, cell(s) must be identified first using the **'Column** and **'Row** properties. See **Grid Property Access**, p.734.

The complete list of properties available for manipulating a grid are described in **Chapter 7. Control Object Properties**, p.705.

Cell Types

The following cell types can be assigned using the **FMT=** option in the **GRID** directive or via the **'Celltype\$** grid object property:

"Button"	Cell works like a button if clicked. Cell can be changed if not locked (on double click). Set 'Lock property to 1 to achieve standard button behaviour for cell.
"CheckBox"	Check box with no 3D effect ('Value=0 or 1,'Text=label).
"CheckBoxRaised"	3D check box that looks raised.
"CheckBoxRecessed"	3D check box that looks recessed.
"CheckMark"	Check box that uses a check mark.
"CheckMarkRaised"	Raised check box that uses a check mark.
"CheckMarkRecessed"	Recessed check box that uses a check mark.
"DropBox"	Text cell that, when editing, allows user to make selection from a drop down list. The drop list is loaded from the 'Text\$ property.
"DropBoxHideBtn"	Same as "DropBox", but the drop down button is shown only when focus is on the cell.
"Ellipsis"	Normal text cell that will show "..." if text exceeds displayable area. Ctrl-Enter can be used to force a new line in the cell (inserts \$0D0A\$ into cell value).
"EllipsisDrop"	Same as "Ellipsis", but forces the cell to drop vertically only during an edit. Ctrl-Enter can be used to force a new line in the cell (inserts \$0D0A\$ into cell value).
"Lookup"	Small button at the right side of cell used to invoke lookup - button image is defined via the cell's 'Bitmap property. If no image is given, button shows three dots.
"LookupHideBtn"	Same as "Lookup", but indicates that the button is hidden when the cell is not selected.
"Multi_line"	Normal text cell that can contain multiple lines of text. Ctrl-Enter can be used to force new line (value \$0D0A\$).
"Normal"	Normal text cell containing one line of data.
"Query"	Same as "Lookup", but user is unable to edit the cell.

"QueryHideBtn"	Same as "LookupHideBtn", but user is unable to edit the cell.
"SingleLine"	Normal text cell that allows for a single line of entry, expanding the cell horizontally according to the overlap rules. It does not allow vertical expansion, like a multiline.
"UseTextNormal"	Similar to "Normal", "SingleLine" and "Ellipsis" respectively, except value of 'Text\$' property is displayed. Edit the 'Value\$' property to edit the cell.
"UseTextSingleLine"	
"UseTextEllipsis"	
"VarDropBox"	Text cell that, when editing, allows user to make a selection from a drop-down list or enter any other value. Drop list selections are loaded using 'Text\$' property.
"VarDropBoxHideBtn"	Same as "VarDropBox", but the drop down button is shown only when focus is on the cell.

Scroll Modes

The **'AutoTrack'** property allows you to control the behaviour of vertical and horizontal scrolling on a grid. The following modes are available:

Normal	Dragging the scrollbar thumb (knob) does not update the grid display until it is released. This is the default mode for scrollbars in ProvideX. It offers the quickest way to scroll through large amounts of data, but lacks the visual reference provided by <i>Scrolltracking</i> (below).
Scrolltracking	Dragging the scrollbar thumb (knob) continuously updates the grid display as it is being moved. This mode makes it easier to see the data relative to the thumb location, but scroll movement may be sluggish depending on how large the data is.
Joystick	This mode works similar to a joystick. It scrolls by one row at a time and the thumb/knob always returns to the center of the scrollbar once it is released from dragging. (Supported for vertical scrolling only).

The following table lists the available **'AutoTrack'** property codes and their associated *horizontal-vertical* scroll modes (*Normal=N*, *Scrolltracking=S*, and *Joystick=J*):

Mode	HScroll	VScroll
0	N	N
1	N	S
2	S	N
3	S	S
4	N	J
5	S	J



Note: Normal and scrolltracking modes would normally be used when the number of rows in the data source is known. Use *Joystick* mode for when the number of rows in the data source is not known.

Format 1: *Define/Create*

GRID *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]

Use this format to create your grid. The value in *ctl_id* is a unique identifier for your grid. This value is used to generate a CTL value whenever the grid is selected and changed.

Format 2: *Remove Grid*

GRID REMOVE *ctl_id*[,ERR=*stmtref*]

Use this format to delete an entire grid.

Format 3: *Disable/Enable*

GRID {DISABLE | ENABLE} *ctl_id*[,ERR=*stmtref*]

Use the **GRID DISABLE** format to disable a grid so that it will be visible but *inaccessible* to users. To reactivate it, use **GRID ENABLE**.

Format 4: *Lock/Unlock*

GRID {LOCK | UNLOCK} *ctl_id,col,row[,width,height]*[,ERR=*stmtref*]

Use the **GRID LOCK** format to allow /deny access to a range of cells in the grid. The user cannot change the cell's value, but if the cell is a button control the user can still click on it. Use the **GRID UNLOCK** format to release the locked range of cells for other processes.

Reminder: If you use zero (0) as your value for *both* column and row, ProvideX interprets that as *all columns and rows, respectively* and will *lock/unlock* the entire grid.

You can also use a grid control object's *properties* to lock or unlock cells (by setting the 'Row, 'Column and 'Lock properties). Set your 'Lock value to 1 (lock) or 0 (zero to unlock). The default background colour for locked cells is LIGHT GRAY (standard button face colour). Use the 'BackColour property to override this. See [Grid Properties, p.146](#).



Note: The current functionality for the 'Sort and 'Sort\$ properties will override locked rows.

Format 5: *Hide/Show*

GRID {HIDE | SHOW} *ctl_id*[,ERR=*stmtref*]

With the **GRID HIDE** format, the grid remains active, but is not displayed. It is still accessible programmatically. Use the **SHOW** format to restore the display and user access.

Format 6: *Force Focus*

GRID GOTO *ctl_id*[,**ERR=stmtref**]

Use **GRID GOTO** directive (without *col,row*) to reactivate and force focus to the *first* cell of your grid, ready for the next user action.

GRID GOTO *ctl_id,col,row*[,**ERR=stmtref**]

Use this format to go to a specific cell address (other than the first cell) and set the focus on that cell, ready for the user's next action. You must include *both* column and row coordinates in this format of the directive (not zero values).

Format 7: *Signal on Focus*

GRID SET_FOCUS *ctl_id,ctl_val*[,**ERR=stmtref**]

Use the **GRID SET_FOCUS** format to define an alternate CTL value to generate whenever focus shifts to the grid.

Format 8: *Add Row/Column*

GRID ADD *ctl_id,col,row*[,**ERR=stmtref**]

Use the **GRID ADD** format to add a row or column of blank cells to your grid. At least one of the *col,row* values must be other than 0 *zero*.

Format 9: *Load*

GRID LOAD *ctl_id,col,row,contents\$*[,**ERR=stmtref**]

Use **GRID LOAD** to load *contents\$* into the grid from a string variable or expression. You must include both the column and the row for the address at which the load is to start. If column and row are both 0 *zero* then the grid is cleared first.

ProvideX can load a number of columns and rows of data at a time. Column and row separators are *mandatory*. Set the column separator using the **SEP=** option. Set your row separator by including it as the last character in your data string. (ProvideX treats the last character in the string as your row separator.)

Adding a New (Bottom) Row: Use *row=0* to add a new row to the bottom of a grid; i.e., `GRID LOAD ctl,1,0,data$+ESC !` (Using ESC as the row delimiter)

Format 10: *Delete Col/Row*

GRID DELETE *ctl_id,col,row*[,**ERR=stmtref**]

Use the **GRID DELETE** format to delete a column or row from your grid. You can identify a column to delete, a row to delete, or both.

Reminder: If you use zero (0) as your value for *both* column and row, ProvideX interprets that as *all columns and rows, respectively* and will *delete* the entire grid.

Format 11: Clear

GRID CLEAR *ctl_id,col,row[,width,height][,ERR=stmtref]*


Use the **GRID CLEAR** format to clear or reset the contents of a range of cells to the default (empty). You can clear a column or a row or both.

Reminder: If you use zero (0) as your value for *both* column and row, ProvideX interprets that as *all columns and rows, respectively* and will *clear* the entire grid.

Format 12: Read First Cell

GRID READ *ctl_id,col,row,var\$,eom\$[,ERR=stmtref]*

Use *var\$* in the **GRID READ** format to receive the modified cell address and value of the *first* selected cell. If the cell doesn't exist, an Error #2: END-OF-FILE on read or File full on write. If there is more to read, the CTL event is re-posted.

The *eom\$* string tells you which input sequence ended the message (e.g., \$0D\$ for the  key).


GRID READ directive must include both column and row addresses. ProvideX returns a queue of cells that have been changed and generates an additional CTL event after a read to ensure that no events are lost. There can be a problem due to potential race conditions when the host is unable to keep up with the user input. To circumvent this, ProvideX returns Error #2: END-OF-FILE on read or File full on write if the queue is empty



Note: Under NOMADS, the read is automatic and should *not* be done by your application. The data read is placed in the standard input variable with *control_name.row* and *control_name.col* containing the associated cell information.

Format 13: Read Next

GRID READ NEXT *ctl_id,col,row,var\$,eom\$[,ERR=stmtref]*

Use the string *var\$* in the **GRID READ NEXT** format to receive the modified cell address and value of the *next* (not first) selected cell. If this cell doesn't exist, ProvideX returns an Error #2: END-OF-FILE on read or File full on write. If there is more to read, no CTL event is re-posted. The CTL value is posted when another cell is changed. The *eom\$* string tells you what input sequence ended the message (e.g., \$0D\$ for the  key)

Formats 14, 15, 16, 17: Select Read/Reset

Use the following formats to select ranges of cells to **READ** or to **RESET** (to remove cells from the **SELECT** state). **GRID SELECT** can be used to control cell selection. (When you **SELECT** a range of cells, the given cells are added to the list of selected cells.)

To determine which cells are selected: use **GRID SELECT READ** to obtain the first selected cell, then use **GRID SELECT READ NEXT** to get subsequent cells.

Reminder: If you use a zero (0) in *both* *col* and *row*, ProvideX interprets that to mean "all cells" and will *select* the entire grid.

GRID SELECT *ctl_id,col,row[,width,height][,ERR=stmtref]*

Select Range. This format selects a range of cells from your grid.

GRID SELECT READ *ctl_id,col,row[,ERR=stmtref]*

Select Read. Use this to read the *first* selected cell. If the cell doesn't exist, ProvideX returns Error #11: Record not found or Duplicate key on write.

GRID SELECT READ NEXT *ctl_id,col,row[,ERR=stmtref]*

Select Read Next. Use this format to read the *next* (not the first) selected cell. When ProvideX encounters a **GRID SELECT READ NEXT** after the last selected cell is returned, it reports

Error #2: END-OF-FILE on read or File full on write.

GRID SELECT RESET *ctl_id[,ERR=stmtref]*

Select Reset. This format removes *all* cells from the **SELECT** state. There is no way to remove cells from the selected state individually.

Format 18: *Find or Retrieve*

GRID FIND *ctl_id,col,row,var\$[,ERR=stmtref]*

Use a string *var\$* in the **GRID FIND** format to return the value for a specific cell.

Format 19: *Write Setting*

GRID WRITE *ctl_id,col,row,contents\$[,ERR=stmtref]*

Use **GRID WRITE** to update your grid by writing settings to specified cells.

Reminder: If zero (0) is used for *both* *col* and *row*, then the value specified in *contents\$* will be written to each cell in the grid.

Format 20: *Report all Changes*

GRID AUTO *ctl_id[,ERR=stmtref]*

Use this format to have ProvideX return a CTL value whenever the current selection is changed. You can use the values in your applications to track a users' selections.

Example

The example on the following page demonstrates how you can use a combination of **GRID** directives and **GRID** control object properties to perform various functions (formatting, loading, retrieving, etc.).

```

0010 ! GRIDDEMO - Grid demo program
0020 BEGIN ;
0020:PRINT 'CS',"Grid Demonstration"
0030 GRID 10,@(3,3,70,15)
0040 GRID LOAD 10,0,0,""
0050 FOR R=1 TO 10
0060 LET R$=""
0070 FOR C=1 TO 10
0080 LET R$+=STR(C*R)+SEP
0090 NEXT
0100 GRID LOAD 10,1,0,R$
0110 NEXT
0120 LET X=10
0130 FOR I=1 TO 10;
0130:LET X'ROW=-1,X'COLUMN=I;
0130:LET X'VALUE$="Col "+STR(I);
0130:NEXT
0140 FOR I=1 TO 10;
0140:LET X'ROW=I,X'COLUMN=-1;
0140:LET X'VALUE$="Row "+STR(I);
0140:NEXT
0150 LET X'COLUMN=-1,X'COLUMNWIDTH=6
0160 LET X'COLUMN=2,X'ROW=0,X'LOCK=1
0170 LET X'COLUMN=3,X'ROW=0,X'COLUMNWIDTH=7,X'CELLTYPE$="Button",X'LOCK=1
0180 LET X'COLUMN=4,X'ROW=0,X'COLUMNWIDTH=7,X'CELLTYPE$="CheckMarkRecessed"
0190 LET X'COLUMN=5,X'ROW=0,X'COLUMNWIDTH=7,X'CELLTYPE$="DropBox",
0190:X'TEXT$="car/pig/dog/"
0200 LET X'COLUMN=6,X'ROW=0,X'COLUMNWIDTH=7,X'CELLTYPE$="DropBoxHideBtn",
0200:X'TEXT$="car/pig/dog/"
0210 LET X'COLUMN=7,X'ROW=0,X'COLUMNWIDTH=7,X'BITMAP$="!bug"
0220 LET X'COLUMN=8,X'ROW=0,X'ALIGN$="C"
0230 LET X'COLUMN=9,X'ROW=0,X'ALIGN$="R"
0240 LET X'COLUMN=0,X'ROW=2,X'BACKCOLOUR$="LIGHT CYAN"
0250 LET X'COLUMN=0,X'ROW=3,X'BACKCOLOUR$="LIGHT YELLOW"
0260 LET X'ROW=8;
0260:FOR I=1 TO 10;
0260:LET X'COLUMN=I;
0260:LET X'BACKCOLOUR$="RGB:"+STR(RND(200)+55)+" "+STR(RND(200)+55)+
0260:" "+STR(RND(200)+55);
0260:NEXT
0270 LET X'ROW=9;
0270:FOR I=1 TO 10;
0270:LET X'COLUMN=I;
0270:LET X'TEXTCOLOUR$="RGB:"+STR(RND(200)+55)+" "+STR(RND(200)+55)+
0270:" "+STR(RND(200)+55);
0270:NEXT
0280 FOR I=1 TO 10;
0280:LET X'COLUMN=I,X'ROW=I,X'FONT$="Arial,1,BI";
0280:NEXT
0290 INPUT X$;
0290:IF CTL=4
0290:THEN STOP
0300 IF CTL<>10
0300:THEN PRINT "Recv'd CTL=",CTL;
0300:GOTO 0290
0310 GRID READ 10,C,R,ZZ$,E$
0320 PRINT @(0,20),'CL',"Col=",C," Row=",R," Dta=",ZZ$," Eom=$",HTA(E$),"$",
0330 GOTO 0290

```


H_SCROLLBAR Directive

Control Horizontal Scrollbar

Formats

1. *Define/Create*: **H_SCROLLBAR** *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]
2. *Define at Edge of Window*: **H_SCROLLBAR** *ctl_id* WINDOW[,ctrlopt]
3. *Remove*: **H_SCROLLBAR REMOVE** *ctl_id*[,ERR=stmtref]
4. *Disable/Enable*: **H_SCROLLBAR** {DISABLE | ENABLE} *ctl_id* [,ERR=stmtref]
5. *Hide/Show*: **H_SCROLLBAR** {HIDE | SHOW} *ctl_id* [,ERR=stmtref]
6. *Force Focus*: **H_SCROLLBAR GOTO** *ctl_id* [,ERR=stmtref]
7. *Read*: **H_SCROLLBAR READ** *ctl_id*,setting,max[,rgn_chg][,arrow_chg][,ERR=stmtref]
8. *Update*: **H_SCROLLBAR WRITE** *ctl_id*,marker,max[,ERR=stmtref]

Where:

- @(col,ln,wth,ht)** Position and size of the horizontal scrollbar region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.
- arrow_chg** Amount to increase/decrease the **H_SCROLLBAR** setting when the user selects the *arrow* at the left/right edge of the horizontal scrollbar. Numeric expression. (Default: 1)
- ctl_id** Unique logical identifier for a horizontal scrollbar (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the **F4** key) or **Negative CTL Definitions, p.817**. Use this value with the apostrophe operator to access various **Horizontal Scrollbar Properties**.
- ctrlopt** Control options. Supported options for **H_SCROLLBAR** include
ERR=stmtref Error transfer
OWN=name\$ Name assigned for automated testing of this control.
OPT=char\$ Attribute/behaviour settings:
 "D" - *Disabled*. User cannot access the scroll bar.
 "G" - *Global*. Keep active on focus change to new/non-concurrent window.
 "H" - *Hide*. Do not display the scroll bar.
 "A" - *Auto*. Generate CTL signal for each movement.
 "S" - *Scroll*. Allow scroll within resizable/scrollable dialogue box.
 Some characters may be combined. Invalid settings are ignored.
- marker** **H_SCROLLBAR** *setting*. Numeric expression between 1 and the maximum, *max*.
- max** Logical maximum value of the **H_SCROLLBAR**. Numeric expression.
- rgn_chg** Amount to increase/decrease the **H_SCROLLBAR** setting when the user selects the *region* left/right of the *marker*. Numeric expression. (Default: max width.)

setting Numeric variable to receive the current scrollbar setting.
stmtref Program line number or label to transfer control to.

Description Use the **H_SCROLLBAR** directive to create a horizontal scrollbar for a window. Your program logic can read and adjust a value by increments to control logical column position within a record every time the user moves the horizontal scrollbar control object.

Horizontal Scrollbar Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a horizontal scrollbar are described in [Chapter 7. Control Object Properties, p.708](#).

Format 1: *Define/Create*

H_SCROLLBAR *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]

When you create the **H_SCROLLBAR**, the value in *ctl_id* is the unique identifier for it. This value is used to generate a CTL value whenever a user selects and moves the horizontal scrollbar; e.g.,

```
0010 H_SCROLLBAR 10000,@(5,20,70,1)
```

The example above defines a scrollbar 70 columns wide and one line high, starting at column 5 on line 20. Whenever the horizontal scrollbar is selected a CTL value 10000 will be generated.



Note: Once a new position is selected, you must read it before your application can use the value to update the actual horizontal scrollbar position.

Format 2: *Define at Edge of Window*

H_SCROLLBAR *ctl_id* **WINDOW**[,ctrlopt]

Scrollbars can appear either within the current window or at the edge of the window. Use the **WINDOW** keyword to have your horizontal scrollbar appear at the edge of the window.

Format 3: *Remove*

H_SCROLLBAR REMOVE *ctl_id*[,ERR=*stmtref*]

Use the **H_SCROLLBAR REMOVE** format to delete the horizontal scrollbar.

Format 4: *Disable/Enable*

H_SCROLLBAR {**DISABLE** | **ENABLE**} *ctl_id* [,ERR=*stmtref*]

Use the **H_SCROLLBAR DISABLE** format to gray out a horizontal scrollbar so that it will be visible but *inaccessible* to users. To reactivate it, use **H_SCROLLBAR ENABLE**.

Format 5: Hide/Show

H_SCROLLBAR {HIDE | SHOW} ctl_id[,ERR=stmtref]

With the **H_SCROLLBAR HIDE** format, the scrollbar remains active, but is not displayed. It is still accessible programmatically. Use the **SHOW** format to restore the display and user access.

Format 6: Force Focus

H_SCROLLBAR GOTO ctl_id[,ERR=stmtref]

Use the **H_SCROLLBAR GOTO** format to reactivate and force focus to a horizontal scrollbar, ready for the next user action.

Format 7: Read

H_SCROLLBAR READ ctl_id,setting,max,[rgn_chg][,arrow_chg][,ERR=stmtref]

Use the **H_SCROLLBAR READ** format to read the **H_SCROLLBAR** settings. For example, to read the horizontal scrollbar position relative to 1000:

```
0120 H_SCROLLBAR READ 100,X,1000
```

Format 8: Update

H_SCROLLBAR WRITE ctl_id,marker,max[,ERR=stmtref]

Use the **H_SCROLLBAR WRITE** format to update or write the **H_SCROLLBAR** settings.

See Also [V_SCROLLBAR Control Vertical Scrollbar, p.365](#),
[Chapter 7. Control Object Properties, p.701](#).

Examples

```
0100 ! 100 - Horizontal Scroll Bar Example

0110 LET VAL=1,MX=400,BJMP=25,SJMP=1
0120 H_SCROLLBAR 101,@(5,16,18,1)
0130 H_SCROLLBAR 102,@(5,18,45,2)
0140 H_SCROLLBAR 103,@(5,21,60,3)
0150 H_SCROLLBAR 104 WINDOW
0160 INPUT (0,HLP="H_SCROLLBAR")@(60,18),"Select...: ','CL',X$
0170 IF CTL<101 OR CTL>104 THEN GOTO 0210
0180 H_SCROLLBAR READ CTL,VAL,MX,BJMP,SJMP
0190 H_SCROLLBAR WRITE CTL,VAL,MX
0200 PRINT @(60,19),"Selection:",CTL,";",STR(VAL),'CL',; GOTO 0160
0210 IF CTL=0 OR CTL>=3 THEN STOP ELSE GOTO 0160
```

HIDE Directive

Hide Control

Format **HIDE** *ctl_id*

Where:

ctl_id Unique numeric CTL identifier for the object (button, list box, etc.) to be hidden. Numeric expression, integer.

Description Use the **HIDE** directive to hide the specified control from display. The control must be either on the current window or on a global control (e.g., *100). It will still be active for purposes of reading/writing but the user cannot see, nor can the program set focus to, the control.

To re-display a control object, use the **SHOW** directive.

See Also [SHOW Show Control, p.326.](#)

Examples

```
0010 BUTTON BTN_VISA,@(10,10,10,2)="&Visa Info"
0020 ....
0100 READ (1,KEY=CST_ID$)IOL=0110
0110 IOLIST CST_TERM$,*,*,INV_DT$
0120 GOSUB 1000
...
1000 IF CST_TERM$="VISA" THEN SHOW BTN.VISA ELSE HIDE BTN.VISA
1010 RETURN
```

IF..THEN..ELSE Directive

Test Condition

Format

1. *Set Conditions:* **IF expression THEN.. [ELSE..] [END_IF or FI]**
2. *Set Conditions, Multiple Lines:* **IF expression THEN {...} [ELSE {...}] [END_IF or FI]**

Where:

- {...}** *Curly braces* indicating that the statements within can span multiple lines. Must be included with both **THEN** and **ELSE** clauses (when used).
- END IF** Optional terminator for the **IF..THEN..ELSE** structure.
- ELSE** Statement executed when *expression* is *false*.
- FI** Optional terminator (identical to **END IF**).
- THEN** Statement executed when *expression* is *true*.
- expression* Numeric expression to be tested for zero.

Description

Use the **IF** directive to control the execution of various ProvideX statements based on the result of a Boolean *expression*. Refer to examples on the following page. If the value returned by the numeric expression is not zero (*true*) then ProvideX continues execution with the directives following the **THEN** clause up to the end of the statement, or until an **ELSE** clause is encountered. If the value returned is 0 zero (*false*), execution of the statement continues with the directives following the **ELSE** clause (if you use it) or with the next statement.

The *expression* would normally include a logical operator (such as an *equals =*, *less-than* symbol *<*, or the **LIKE Operator**, p.822), but you can use any numeric expression.

All statements within an **IF..THEN..ELSE** structure exist on the same line. These statements can only span multiple lines if they are enclosed within *curly brackets*. A matched set of open/closed brackets must be provided for each set of directives (missed brackets can cause unexpected results).

Internally, when the system detects a **{** following the **THEN** clause it will continue execution up to the next **}** (if *true*) or skip forward to it (if *false*). The same holds true for the processing of the **ELSE** clause. Curly braces should only be used with **ELSE** if they are also used with **THEN**. You can imbed multiple levels of multi-lined **IF** directives via curly braces; however, it is important not to lose consistency.

Using END_IF

An optional **END_IF** (or **FI**) clause can be used to terminate the current **IF** structure and/or to execute a common closing statement. This is particularly useful for separating **ELSE** clauses in a nested **IF..THEN..ELSE** structure. Once the statements that follow an **END_IF** clause are executed, control will fall through to the next line, or (if nested) to the preceding level of **IF..THEN..ELSE**.

See Also

- END_IF End IF Directive**, p.117
- SWITCH..CASE Branch Control**, p.331
- LIKE Operator**, p.822
- Decision Structures**, *User's Guide*.

Examples

Simple IF Statement

```

00010 INPUT "Enter a number: ",a
00020 INPUT "Enter another: ",b
00030 IF a>b \
        THEN PRINT "First one is larger";
        STOP
00040 IF b>a \
        THEN PRINT "Second one is larger";
        STOP
00050 PRINT "Both numbers are the same"
00060 STOP
-:RUN
Enter a number: 123.45
Enter another: 123.56
Second one is larger

```

Compound IF Statement:

```

00010 FOR i=1 TO 30
00020 PRINT i," is divisible by ",
00030 IF MOD(i,2)=0 \
        THEN IF MOD(i,3)=0 \
                THEN PRINT "both" \
                ELSE PRINT "2" \
        ELSE IF MOD(i,3)=0 \
                THEN PRINT "3" \
                ELSE PRINT "neither"
00040 NEXT i
00050 STOP
RUN
1 is divisible by neither
2 is divisible by 2
3 is divisible by 3
4 is divisible by 2
...
28 is divisible by 2
29 is divisible by neither
30 is divisible by both

```

Multiple Line IF Statement.

```

00010 IF A = B THEN {
00020     LET C = D
00030     PRINT "A equals B"
00040 } ELSE {
00050     LET X = Y
00060     PRINT "A was not identical to B"
00070 }

```

INDEXED Directive

Create Indexed File

Format	<code>INDEXED filename\$,[max_recs],rec_size[,SEP=char\$][,ERR=stmtref]</code>
	<i>Where:</i>
<i>char\$</i>	Separator character. Hex or ASCII string value.
<i>filename\$</i>	Name of the indexed file to create. String expression.
<i>max_recs</i>	Maximum number of records in the file. Optional numeric expression. Default is no initial allocation of file space, with no limit as to final size. 0 <i>zero</i> indicates that the number is dynamic. A positive value indicates that the file is pre-sized to the specified number of records – an Error #2: END-OF-FILE on read or File full on write will be generated if an attempt is made to add more than this specified number of records, or access an IND= value above this limit.
<i>rec_size</i>	Maximum size of the data portion of each record.
<i>stmtref</i>	Program line number or label to transfer control to.

Description Use the **INDEXED** directive to create a file that can be accessed either sequentially or by record number (index).

Make the record size long enough to contain the maximum record length (i.e., the combined length of the data fields plus field separators). If you use a filename that already exists, ProvideX returns an Error #12: File does not exist (or already exists).

Use the **SEP=** option to set the default separator for a given file. Use any character from \$00\$ to \$FF\$, or use `SEP=*`. When the *asterisk* * is set as the value, fields will not be delimited. ProvideX writes records to the file with field type and length headers. This feature may provide better results in overall file performance.

WindX supports the use of this directive via the [WDX] tag; e.g., `INDEXED "[WDX]somefile.ext" ...` For more information, see [\[WDX\] Direct Action to Client Machine, p.801](#).

See Also [File Types, User's Guide](#).

Examples

```
0110 INDEXED A$+"-"+B$,100,50,ERR=1090
0200 INDEXED "CSTFLE",,140
```

Line 0200 creates a file with the following structure:

```
Indexed file: C:\Program Files\Sage Software\ProvideX\CST\CSTFLE
Maximum Record size .....: 140
Maximum # records .....: (No limit)
Current # records .....: 0
```

INPUT Directive

Get Input from Terminal

Format **INPUT** [**EDIT**] (*[chan][,fileopt]*)*varlist*

Where:

chan Channel or logical file number of the file from which to get terminal input.

fileopt Supported file options (see also, [File Options, p.810](#)):

ERR=*stmtref* Error transfer

HLP=*string*\$ Help message identifier

IND=*num* Position cursor to specified column number.

LEN=*num* Limit on input size

SIZ=*num* Number of characters to read (number of screen columns)

TBL=*stmtref* Data translation table

TIM=*num* Maximum time-out value in integer seconds.

When screen positions are tight, you can combine the **LEN=** and **SIZ=** options to have ProvideX supply a scrolling input field. For example, to allow a user to enter 60 characters in a field where you only have room for 30 on the screen: `INPUT (0,LEN=60,SIZ=30)"Name . . . : ",N$`

varlist Comma-separated list of variables, literals, [Mnemonics](#), **IOL=** options, and/or location functions '@(...)'. Include [Data Format Masks](#) to filter data being received.

Description Use the **INPUT** directive to issue prompts to and receive input from a terminal device. You would normally use the logical file number to refer to a terminal. Input from the user is stored in a variable specified. ProvideX treats any literals or expressions included in the statement as prompts. When a numeric variable is specified, numeric data must be received. Non-numeric input in response to a numeric variable (other than commas and decimals) will cause an `Error #26: Variable type invalid`.

If you use the **EDIT** clause, the current values for the variables are loaded into the input buffer so that the user can edit them.

Use a format mask with the **INPUT** directive to control the input. To do this, append a colon and the mask to the given variable in *varlist*. If you omit a format mask for a numeric in an **INPUT** statement, the **'DP' Decimal Point Symbol** and **'TH' Thousands Separator** system parameters are ignored for European decimal settings.

If an **INPUT EDIT** statement has both a format mask and a validation list, make sure that the validation list contains only entries that are possible. For example, if the format mask indicates that the input value must be a single character, do not use a null ("") value in the validation list.

Instead, valid input for "no value" could be a blank / space in this instance, as in:

```
0030 INPUT EDIT (0,ERR=0030)@(10,10),VAR$:"A":("Y"=0040,"N"=0050," "=0060)
```


Use the **IND=** option to set the position of the cursor in the **INPUT** field. Use **IND(chan)** to obtain the position where the cursor was left after the **INPUT**.

See Also

ACCEPT Read Single Keystroke, p.28
OBTAIN Get Hidden Terminal Input, p.227
'ME' Mnemonic, p.620
'BI' Mnemonic, p.590
Data Format Masks, p.813
Input Statements, *User's Guide*.

Examples

```
0010 INPUT 'CS',@(5,5),"Enter customer number:",C ...
0100 INPUT EDIT "Enter value:",INV_AMT:"$##,##0.00"
```

Use **IND=X** as a parameter for the **INPUT** statement to set the starting point in an input field. For example, to start the input at the tenth character of **A\$**:

```
-> A$="Now is the time"
-> INPUT EDIT (0,IND=10) A$
```

Use **IND(0)** to find out the character position where the user terminated input. For example, if the user enters "Now is the time", and presses **Home** and **Tab** the cursor moves to the 10th position, just after the "e" in "the". Then, if the user presses **Enter** or any other function key to terminate the input, the value returned in **IND(0)** is 10.

INSERT Directive*Insert New Record in File*

Formats **INSERT** (*chan*[,*fileopt*])*varlist*

Where:

chan Channel or logical file number of the file to which to write.

fileopt Supported file options (see also, [File Options, p.810](#)):
BSY=*stmtref* Traps Error #0: Record/file busy
DOM=*stmtref* Missing record transfer
END=*stmtref* END-OF-FILE transfer
ERR=*stmtref* Error transfer
IND=*num* Record index
KEY=*string*\$ Record key
REC=*name*\$ Record prefix (**REC=VIS(*string*\$)** can also be used)
RTY=*num* Number of retries (one second intervals)
TIM=*num* Maximum time-out value (support write operations for TCP channels).

stmtref Program line number or label to transfer control to.

varlist Comma-separated list of variables, literals, and **IOL=** options.

Description The **INSERT** directive is used to write a new record to a file (channel/logical file number). The syntax for this directive is identical to the [WRITE Directive, p.383](#); however, **INSERT** only writes a record if *does not exist* and will return an error if the record already exists.

INSERT may be used against Keyed, Memory, ODBC, and OCI files. When **IND=** is used with ***MEMORY***, this directive inserts a new index.

See Also [WRITE RECORD Write Record, p.386](#)
[WRITE Add/Update Data in File, p.383](#)
[UPDATE Update Existing Record in File, p.351](#)

INVOKE Directive *Execute Operating System Command*

Formats

INVOKE [HIDE] [WAIT] [CONTROL] *command\$* [,ERR=*stmtref*]

Where:

command\$ String expression to be passed to the operating system command processor for execution.

CONTROL Optional keyword allows a program to invoke an external program on Vista with the Run as Administrator option. (*Windows-only*)

HIDE Optional keyword will minimize the invoked task when it starts. (*Windows-only*)

WAIT Optional keyword causes the ProvideX session to wait until the application terminates or the user reactivates the task. (*Windows-only*)

stmtref Program line number or label to transfer control to.



Note: In Command mode, ProvideX interprets a statement as an **INVOKE** directive if it begins with an double quotation mark (").

Description

The **INVOKE** directive enables a statement in a program to be processed by the operating system as if it were entered as a command from the terminal. It provides access to most operating system utilities and facilities from within a ProvideX program.

In *Execution mode*, character strings are passed from a program to the operating system. In *Command mode*, entering a line of code that starts with a quotation mark (i.e., "statement ") is the same as using the word **INVOKE** to start the directive.

In *Windows*, to accommodate the way Windows searches for programs and determines the type of program to run, it is best to identify the type of file in the command line (e.g., "XCOPY . EXE" or "MYJOB . BAT"). Under *WindX*, use **INVOKE** "[WDX] . . . ". See [\[WDX\] Direct Action to Client Machine, p.801](#). If the operating system command does not use a window when invoked, ProvideX will not wait for the command to finish as there is no focus change to wait for.



Tip: Use the **SYS()** function instead of the **INVOKE** directive if you wish to test the successful conclusion of an operating system command.

If you are not concerned about the return status of a command and are running PVX UNIX and PVX Windows, there is no difference between using an **INVOKE** directive and the **SYS()** function. When **SYS()** is used under Windows, ProvideX waits for the spawned task to complete or for the program to be reactivated via the mouse. When the **INVOKE** directive is used under Windows, ProvideX spawns the task and continues without waiting for the task to complete.

Use **INVOKE WAIT** directive to wait for the task, which is the same as using the **SYS()** function. Use **INVOKE HIDE** to spawn a task in a minimized window.

INVOKE CONTROL launches the task on Windows Vista specifying Run As Administrator. Also, if **INVOKE WAIT** is used on Windows Vista, ProvideX will first try to launch without requesting Administrator rights. If the launch fails with `ERROR_ELEVATION_REQUIRED`, ProvideX will retry the launch with the "Run As Administrator" option.

The Windows options **WAIT**, **CONTROL**, and **HIDE**, are simply ignored by ProvideX if used under UNIX/Linux.

The **INVOKE** directive can be also used to have the operating system treat a character string from a program as an operating system command from the terminal; e.g.,

```
0010 INVOKE ENV("COMSPEC")+ " /K DIR C:\WINDOWS"
```

The `ENV("COMSPEC")` function determines the correct command processor to be used.

If the OS command fails on an **INVOKE**, the program will continue processing unless an **ERR=** clause was included in the statement. Generic error traps, such as **SETERR** or **ERROR_HANDLER** do not get triggered by **INVOKE** related errors.

Background Tasks in ProvideX

Use **INVOKE** to start a background process in ProvideX.

Under UNIX, output generated by the OS command may result in strange display behaviour. In particular, ProvideX will not know where the cursor is, nor be able to restore the screen if a window is pushed/popped. This is because the output from the OS command is not intercepted by ProvideX and is therefore unknown to it.

When you **INVOKE** another copy of ProvideX under UNIX, we recommend that you also redirect `stdin`, `stdout`, and `stderr` to null files; e.g.,

```
INVOKE ARG(0)+"program >/dev/null 2>/dev/null </dev/null"
```



Note: `ARG(0)` returns the name of the 'pvx' executive as entered in the original command line. Please make sure that this is an absolute pathname or that ProvideX can be found in the system path search rules.

See Also

[SYS\(\) Function, p.529.](#)

IOLIST Directive

Specify Variable List

Formats

1. *Define List*: **IOLIST** {*parm*[:*format*]...}

Note: The inner set of brackets enclosing [*format*] are part of the syntax.

Where:

parm Set of parameters to be used in an input/output directive. This includes variables, literals, **Mnemonics**, **IOL=** options, and/or location functions '@(...)'.
format Defines how the field is formatted. Options include:

CHR(*len*) Variable length string (pad output).

CHR(*d/m*) Variable length string (delimited). Default record limit is 10240 for composite IOLists with the **CHR** field definition.

DEC(**[SEP,SIZ=]***len*,*scl*) Fixed length, explicit decimal, sign maintained. Optional **SEP,SIZ=** clause allows for padded numeric fields followed by a field separator. Numeric fields are padded with leading spaces.

LEN(*len*) Fixed length string.

STR(*d/m*) Quoted String.

NUM(*len*,*scl*)

Fixed length numeric.

BIN(*len*,*scl*) Binary numeric.

INT(*len*,*scl*) Unsigned integer numeric.

BCD(*len*,*scl*) Packed decimal numeric.

Where:

len is the length of the field in the record.

d/m is the string whose first character delimits the field.

scl is the optional scaling factor to apply to the number (e.g., 2 indicates that 1.34 is output as 134).

Description

Use **IOLIST** to define a common I/O parameter list (IOLIST). ProvideX ignores the **IOLIST** directive when it encounters it during program execution until it is used in conjunction with file I/O directives (e.g., **READ**, **WRITE**, etc.) and composite string definitions. For more information, see *File Handling* in the *ProvideX User's Guide*.



Note: **IOLIST** must be the only directive in a statement.

Examples

```
0010 OPEN (20)"CSTFLE"
0020 READ (20,KEY=C$)IOL=1000
0030 LET J=J+10
0040 WRITE (20,KEY=C$)IOL=1000
0050 WRITE (21)IOL=1010
...
1000 IOLIST C$,C1$,C2$,C3$,J,K,L,M
1010 IOLIST C$:[CHR(10)],C1$:[CHR(40)]
```

KEYED Directive Create Single/Multi-Keyed File

Format

KEYED *filename* \$ [, *extkey_len*] [, *key_def* \$] [, *max_recs* [, *rec_size*]] [, *fileopt*]

Where:

filename \$ Name of the Keyed file to create. String expression.

extkey_len Numeric expression. Length of the external key for all records in the file (maximum 127 characters).

key_def \$ String expression defining the key. The Keyed file can be single- or multi-keyed. A key definition is made up of one or more key field definitions ranging 0 to 15 for FLR/VLR files or 0 to 255 for EFF files. Use integers for specific field numbers, or 0 *zero* for record-based offsets. The key definition formats are as follows:

Single key field:

```
[["keyname":]field:offset:len[:"attr" ]
```

Composite key fields (using the + operator):

```
[["keyname":]field:offset:len[:"attr" ]]+[field:offset:len[:"attr" ]]
```

Multi-keyed alternate key fields are comma-delimited:

```
[["keyname":]field:offset:len[:"attr" ]], [["keyname":]field:offset:len[:"attr" ]]
```

Where:

keyname Name of key assigned for use in **KNO=name** \$ options.

field Integer, 1 to number of fields (0 = record-based offset).

offset Starting position within the field (integer, 1 to 3839).

len Number of characters in the key field (integer, max. 127)

"attr" Attribute characters, see [Key Definition Attributes, p. 167](#)

: Colon - the separator for elements in a key segment.

The maximum total size for an external key is 127 characters. The maximum length for internal or alternate keys is 240 characters.



Note: The *outer* set of square brackets in the above formats are part of the syntax; the *inner* brackets indicate optional syntax items (i.e., the brackets enclosing the optional [:"attr"] are not part of the syntax).

max_recs Maximum number of records the file is allowed. Optional numeric expression. The default is zero (no limit). (Use a comma with no value to set the default.) If a positive value is supplied, ProvideX creates and pre-allocates disk space for the file. With a negative value, ProvideX allocates sufficient disk space for the file, but will set the *max_recs* count back to zero (unlimited).

rec_size Maximum size of the data portion of the record (excluding the key). A negative value creates a variable-length record (VLR) data file with the maximum record length equal to the positive value of this field. A positive value creates a fixed-length record (FLR) formatted file.

If you do not specify size, the default is VLR with a maximum record size of 256. The maximum block size for a VLR file is 31KB and the maximum record size is 31000 bytes. Attempting to create a VLR file with a record size more than 31000 bytes results in an FLR file with the requested record size.

fileopt

Supported file options (see also, [File Options, p.810](#)):

BSZ=*num* Block size. Numeric expression (1 - 63).

ERR=*stmtref* Error transfer.

SEP=*char*\$ Default field separator character. Hex or ASCII string value.

OPT=*char*\$ Single character settings:

"C" - *Compressed*. Adds simple compression to record data.

"X" - *Extended Record Size*. Extends record sizes up to 2GB per record.

"0" - *Create VLR/FLR files (default if 'XF'=0)*

"1" - *Create EFF Files with 2GB limit*.

"2" - *Create EFF Files without 2GB limit (supported platforms)*.

"Z" - *Set ZLib Compression for VLR and EFF Files*.



Note: OPT="2" generates Error #99: Feature not supported on platforms that do not offer Large File Support (LFS). Using options "Z" and "C" together will result in an Error #32.

Key Definition Attributes

Optional key definition attribute characters "*attr*" can be used to refine key segment definitions; e.g., KEYED "MES_FACTURES", [1:1:40:"T"], [2:1:40:"TCD"].



Note: Only one of the auto-increment options ("#", "+" or "I") can be used per file.

Character	Definition
"#"	Auto-increment - Space filled string. (<i>VLR/EFF only</i>)
"+"	Auto-increment - Zero filled string. (<i>VLR/EFF only</i>)
"_"	Support signed binary integers as key segments. This option cannot be used in conjunction with the following segment options on the same segment: "#", "+", "C", "D", "I", "L", "T", "U", "Z".
"A"	Ascending key (<i>assumed</i>).
"C"	Force uppercase for individual key segments to create case-insensitive keys; e.g., KEYED "filename", [1:1:10:"C"], [2:1:40:"L"]+[1:1:10:"C"] Note: The conversion tables for upper/lowercase conversions are located in the ProvideX message file *MLFILE.EN. You can set these tables using the DEF UCS and DEF LCS directives and retrieve values using the UCS(*) and LCS(*) functions.
"D"	Descending order (default is ascending).

Character	Definition
"I"	Binary auto-increment (<i>VLR/EFF only</i>). Field (1, 2, or 4 bytes in length) will be auto-incremented as records are added to the file. Key must be record-based; i.e., field is 0 and offset is location in record.
"K[:n]"	Null key suppression (<i>VLR/EFF only</i>). <i>n</i> is the hex value of the character to suppress if all characters match (defaults to \$00\$ if not specified). If the entire key evaluates to null, the key will not be a part of this key tree. This cannot be used on a primary key or in conjunction with the "N" option.
"L"	Force lowercase for individual key segments to create case-insensitive keys. See the example and note in "C", above.
"N[:n]"	Null segment suppression (<i>VLR/EFF only</i>). <i>n</i> is the hex value of the character to suppress if all characters match. If any segment within the key evaluates to null, the key will not be a part of this key tree. This cannot be used on a primary key or in conjunction with the "K" option.
"S"	Swapped (same as SWP() type 7, Intel x86 style swapping only <i>VLR/EFF only</i>).
"T"	Force automatic accent translation for individual key segments (i.e., to translate accented/extended characters to their non-accented counterparts). You'll find this useful for sorting multilingual characters into a single unified sort sequence; e.g., KEYED "filename", [1:1:40:"T"], [2:1:40:"TCD"] Note: The conversion tables for accent translations are located in the ProvideX message file *MLFILE.EN. You can set this table using the DEF CVS directive and retrieve values using the CVS(*) function.
"U"	If you declare an alternate key segment as <i>unique</i> , then no duplicate keys are allowed on writing a record. (ProvideX generates Error #11: Record not found or Duplicate key on write.) If you designate any segment in a key definition as unique, then the entire key will be unique. Note: The primary key must always be unique.
"Z"	Zero terminated strings (<i>VLR/EFF only</i>). This will consider a null byte (\$00\$) in a string as the end of the data within the field, and will ignore any data between the null byte and the end of the key field. (Also known in C as a Z-string.)

Description Use the **KEYED** directive to create a file with one or more keys. If the first field in the directive after the filename is a number, ProvideX creates an *external* file key (i.e., an index to the file). If the first field in the directive is a key definition enclosed in square brackets [], then ProvideX uses only *internal* key fields instead.

ProvideX considers the first key specified for a Keyed file to be the primary key. Every record must have a unique primary key. You can have duplicate secondary keys from record to record.



Note: By default, the **KEYED** directive will create variable-length record (VLR) data files or fixed-length record (FLR) formatted files; however, by setting the 'KF' system parameter ("KF"=2) or by setting OPT=" 2 " in the syntax, **KEYED** can also be used to create Enhanced File Format (EFF) files on platforms that support Large File System (LFS), 64-bit addressing. For more information on EFF refer to the [CREATE TABLE Directive, p.58](#).

For *VLR/FLR* files, there is a maximum of 16 key fields allowed on a file with a maximum of 96 data components making up the 16 keys. There is no limit (other than the maximum of 96 key components) to the number of fields that comprise a single key. For *EFF*, there is a maximum of 255 keys allowed on a file with a maximum of 255 data components making up the 255 keys. *The initial implementation of EFF (in Version 6) is limited to 96 keys and 96 segments.*

If a given filename already exists, ProvideX returns Error #12: File does not exist (or already exists).

Use the **SEP=** option to set the default separator for a given file. Use any character from \$00\$ to \$FF\$, or use **SEP=***. When the *asterisk* * is your value, the fields will not be delimited. ProvideX writes records to the file with field type and length headers. This feature can provide better results in overall file performance.



Note: WindX supports the use of this directive via the [WDX] tag; e.g., KEYED "[WDX]somefile.ext" ... See [\[WDX\] Direct Action to Client Machine, p.801](#).

For more information refer to [DIRECT Create File with Keyed Access, p.89](#), and *Multi-Keyed Files* in the *ProvideX User's Guide*.

If you see the message `Corrected Missing Key` when ProvideX attempts to **WRITE**, **UPDATE**, or **REMOVE** a record in a Keyed file where a change to the primary or alternate key table is needed (e.g., for an alternate key to a customer name when the name changes or for removal of all keys when a record is deleted), the message indicates that the key entry to be changed was not found. This is not a fatal message, but usually means that the file has been corrupted in some way. You should check the file using *UFAC or try to recover it using *UFAR (ProvideX utilities).

Key Block Size and Performance

ProvideX stores a maximum of 255 keys per block, with block sizes ranging from 1Kb to 32Kb. The maximum key block size for a variable record length file is 31Kb, while the maximum for a fixed record length file is 32Kb and an EFF file is 63Kb. The size of the key block also governs the size of the blocks used to store data records. As with key blocks, the maximum number of records per data block is limited to 255.

By default, ProvideX uses the size of the primary key to determine the optimum key block size to a maximum of 4Kb when creating a file. This calculation involves taking the (size of the primary key + 5 bytes) multiplied by a maximum of 255 entries, then rounding up to the nearest Kb to a maximum of 4Kb. The block size can be overridden by using the **BSZ=** option on the **KEYED** directive. This can be used to optimize the usage of key and/or data blocks to improve performance. For

example, using a `BSZ=1` improves the performance on WANs (Wide Area Networks) by reducing the overall network traffic when accessing Keyed files.

The number of keys stored per block also affects the number of active levels on the key tree. Storing a smaller number of keys per block results in additional levels of blocks on the key tree, whereas a larger number of keys often results in fewer levels. Having fewer levels translates into improved performance as this reduces overall network traffic. For example, a key size of 50 would yield a maximum of 74 keys per block using a 4Kb block size while an 8Kb block size would accommodate 148 keys per block. In a 4K file, three levels on the key tree would be capable of managing 405,224 records ($74 * 74 * 74$) and 3,241,792 records ($148 * 148 * 148$) for an 8K block size.

Keys have the following limits:

- Maximum external key size is 127 bytes.
- Maximum internal or alternate key size is 240 bytes.
- Maximum segment size is 127 bytes.



Note: For VLR and EFF files, the `BSZ=` must be large enough to accommodate the record size; e.g., 4Kb block size for a 4000-byte record. You no longer have to use `BSZ=` in a Keyed file definition if defining record sizes over 4000 bytes. ProvideX now calculates the minimum block size necessary.

Variable Length Records

When using variable length record or EFF files, ProvideX will not pad the record to the record size with extra null characters (as is done with fixed length record files). The record data is stored using its exact length. This allows for much denser files, by avoiding the null padding on each record. Files are smaller thus when doing reads of large portions of the file, less disk accesses have to be performed because the records fit into a much smaller space and disk cache is better utilized.

In addition, with a variable length keyed file, the maximum record size is a logical maximum and not a physical one. A VLR file will simply allow you to write records from 0 to the maximum record size defined. Anything beyond the maximum record size causes an `Error #1: Logical END-OF-RECORD reached`. This design allows the maximum record size for a variable length record file to be increased at any time through the use of system utilities, `*UFAM`.

Only VLR files support file segmentation, which allows you to have files that total a logical size of up to about 248 GB. Refer to the ['MB'= System Parameter, p.674](#).

See Also

[CREATE TABLE Create Keyed File \(EFF\), p.58](#)
[DIRECT Create File with Keyed Access, p.89](#)
[SYSTEM_JRNL File System Journalization, p.334](#)
[ADD INDEX Add Key to Keyed File, p.29](#)
[DROP INDEX Drop Key from Keyed File, p.103](#)
[RENAME..INDEX Rename Keys in Keyed File, p.285](#)
[SORT Create File for Sorting, p.327](#)
[Accessing Data Files, p.22](#)
[File Types, User's Guide](#)

Examples

KEYED A\$+"_"+B\$,10,100,50,ERR=1090 creates a Keyed file with this structure:

```
Maximum Record size .....: 50
Maximum # records .....: 100
Current # records .....: 0
Size of key block .....: 3072 bytes
External key size .....: 10
```

KEYED "CSTFLE1",6,,140 also creates a file with an external primary key (6 bytes).

KEYED "CSTFLE2",6,[1:1:10],,500 creates a file with an external primary key (6 bytes) and a single internal alternate key.

KEYED "CSTFLE3",[1:1:6],[2:1:10],,500 creates a multi-keyed file with 2 internal alternate keys. Below, the alternate key 0 (field 1) is the primary key:

```
Keyed file: C:\OTHER\MANUALS\PVX\CST\CSTFLE3
Maximum Record size .....: 500
Maximum # records .....: (No limit)
Current # records .....: 0
Size of key block .....: 2048 bytes
External key size .....: 0
Alt. key 0 .....: [1:1:6]
Alt. key 1 .....: [2:1:10]
```

KEYED "CSTFLE4",6,[2:1:10]+[1:1:6] creates a file with an external primary key, and a composite internal alternate key:

```
Keyed file: C:\OTHER\MANUALS\PVX\CST\CSTFLE4
Maximum Record size .....: 256 (Variable)
Maximum # records .....: (No limit)
Current # records .....: 0
Size of key block .....: 2048 bytes
Record Expansion factor .....: 10%
External key size .....: 6
Alt. key 1 .....: [2:1:10]+[1:1:6]
```

To use a variable for the key definition:

```
X$="[1:1:6],[2:1:10]"; KEYED "CSTFLE",X$,,500
X$="6,[1:1:6],[2:1:10]"; KEYED "CSTFLE",X$,,500
```

KEYED LOAD Directive *Load and Repair Keyed File*

Format **KEYED LOAD** (*chan*,[**KNO**=*num* | *name*\$_])

Where:

chan Channel or logical file number of the file to be read/repaired.

num Key number (0-15 for VLR/FLR files, 0-255 for EFF files).

name\$_ Name of the key (if assigned). String expression.

Description This directive is used to rebuild the key tables of a file which is currently open on a channel. **KEYED LOAD** (*chan*) rebuilds all key trees. To rebuild a specific key chain, use the file access key value **KNO**=*num* | *name*\$_ **KEYED LOAD** will identify duplicate primary or unique keys.

The file does not have to be locked for **KEYED LOAD** to work. **KEYED LOAD** rebuilds the key chains within the file without using more disk space, while the file is in use, and while the file is being updated by others. **KEYED LOAD** is much faster than *UFAR in repairing key trees.

However, it performs much faster if the file is locked, and if there is sufficient ram available for its key block buffering operations. Also, **KEYED LOAD** will update the number of records for fixed length files if the file is locked.

Once the **KEYED LOAD** command is completed, **TCB(67)** will be set to one of the following values:

- > 0 represents the number of keys reloaded.
- 1 indicates that a different number of keys were encountered on different key chains. This may or may not indicate that the file being re-loaded has physical damage that **KEYED LOAD** is unable to correct; i.e., the file can be reloaded on the fly while others are adding records, and therefore the number of keys on a given key chain may have changed while **KEYED LOAD** was working.



Note: **KEYED LOAD** reports invalid file type when used on Sort files. **KEYED LOAD** will not operate on files opened in Read-Only mode.

Example

```

->open (1)"datafile.dat"
->keyed load (1)
Done - loaded key number 0 with 263 keys
Done - loaded key number 1 with 263 keys
Done - loaded key number 2 with 263 keys
Done - loaded key number 3 with 263 keys
Done - loaded key number 4 with 263 keys
Done - loaded key number 5 with 263 keys

```

LET Directive

Assign Value to Variable

Format `[LET] var[$]=expression,[var[$]=expression,...]`

Where:

var Numeric or string variable to be set to the value of the expression.

expression Numeric or string expression whose value will be assigned to the variable.

Description Use the **LET** directive to set a variable to the value of an expression. If the variable is numeric, use a numeric expression. If it is a string variable, use a string expression. You can use substrings with string variables. (Specify the starting position and optional length.)

The word *LET* is optional in Command mode. ProvideX assumes use of the **LET** directive when it encounters a directive starting with a variable. You can include multiple **LET** directives in one statement by using comma delimiters. In this case, if you use the word *LET*, it only occurs once, before the first assignment.

You can initialize, copy, or manipulate a complete array by specifying **{ALL}** following the variable name. **{ALL}** can be used on both sides of an equation. Note that the curly brackets **{ }** are part of the syntax.

Examples

```
0080 LET A$="THIS IS AN EXAMPLE"
0100 A$(1,4)="WHAT",A$(19)="!",Z$=" "
0110 A$(5,4)=Z$
```

If Z\$ is more than 4 characters, A\$(5,4) is set as the first 4 characters of Z\$. If Z\$ is less than 4 characters, the A\$(5,4) value is Z\$ plus space-padding up to a length of 4 characters:

```
RUN
-: ?A$
WHAT    AN EXAMPLE!
```

Other Examples:

```
-: Z4=A+4.5,Z5=Z4*.85
-: LET D$=DAY,T=TIM*3600
-: DIM A[30]; LET A{ALL}=A{ALL}*4
```

LIKE Directive

Inherit Properties

Format **LIKE** "otherclass", "otherclass", ...

Where:

otherclass Name of a class to inherit properties from.

Description The **LIKE** directive is used in *Object Oriented Programming* (OOP) to inherit the properties from one or more other classes. All properties and methods are inherited from the specified classes.

When multiple occurrences of the same property/function are found within the inheritance, the first class declared in the **LIKE** directive takes precedence.

Example

```
DEF CLASS "Company"
FUNCTION Delete()
REMOVE (fileno, KEY=COMP_ID$)
RETURN 1
END DEF
```

```
DEF CLASS "Client" FUNCTION Delete()
IF AMT_OWING <> 0 RETURN 0
REMOVE (fileno, KEY=CUST_ID$)
RETURN 1
END DEF
```

```
DEF CLASS "Dealer"
LIKE "Client", "Company"
END DEF
```

The Dealer class of objects would use the "Client" 'Delete()' method, since it was declared first.

See Also

[DEF CLASS Define Object Class, p.65](#)
[FUNCTION Declare Object Method, p.137](#)
[LOCAL Designation of Local Data, p.197](#)
[PROGRAM Create/Assign Program File, p.259](#)
[PRECISION Change Current Precision, p.248](#)
[PROPERTY Declare Object Properties, p.261](#)
[Data Integration, User's Guide](#)

LINE_SWITCH Directive *Redirect Console Input/Output*

Formats

1. *Switch File (0)*: **LINE_SWITCH** *chan* [,**ERR=stmtref**]
2. *Switch File (0)*: **LINE_SWITCH** [*device\$*,**ERR=stmtref**]

Where:

- chan* Channel or logical file number to which to redirect file 0 *zero* input and output.
- device\$* Name of the device to which to switch file 0 *zero*. String expression.
- stmtref* Program line number or label to transfer control to.

Description

Use the **LINE_SWITCH** directive to switch all I/O currently sent to channel or logical file number 0 (zero, the terminal) to another device. Its primary purposes are to allow for secondary users on a PC or to provide remote maintenance capability.

While the **LINE_SWITCH** is in effect, the original device is disabled. ProvideX will expect to receive all input from and send all output to the new device designated as file or channel 0 *zero*.

Use a **LINE_SWITCH** directive with no parameters to restore communications with the original device on channel (logical file number) 0 and **CLOSE** the alternate file.

Examples

- ```
-> LINE_SWITCH "PORT2" ! Defines/opens PORT2 as logical file number (0)
-> LINE_SWITCH ! Closes PORT2, restores terminal/console as logical file
 #(0)
```

# LIST Directive

## List Program Statements

### Formats

1. List a Range of Lines: **LIST** [**EDIT**] [(*chan*,*fileopt*)] [*from\_stmt*][,*to\_stmt*]

2. List a Range of Lines: / [**EDIT**] [(*chan*,*fileopt*)] [*from\_stmt*][,*to\_stmt*]

#### Where:

/ or \ Use either of the slashes (forward or back) as a substitute for typing **LIST**. ProvideX also accepts **LSIT** (i.e., mis-typed version).

**EDIT** Use the optional keyword **EDIT** with the **LIST** directive to have ProvideX both logically break the lines and indent them. Refer to the '**LE**' System Parameter, *p.672*, for more information.

*chan* Channel or logical file number to receive the readable listing of the contents of a program.

*fileopt* Supported file options (see also, [File Options, p.810](#)):  
**END=stmtref** End-Of-File transfer  
**ERR=stmtref** Error transfer  
**IND=num** Record index  
**TBL=stmtref** Data translation table.

*from\_stmt* Starting statement to list. Optional. If you omit this, the default is to LIST from the start of the program.

*to\_stmt* Ending statement to list. Optional. If you omit this, the default is to LIST to the end of the program.

*stmtref* Program line number or label to transfer control to.

### Description

The **LIST** directive converts program statements from the internal (compiled) format to a readable listing that can be sent to a file or to the screen. This directive gives you a listing of the contents of a program's statements as they were entered into the system originally. That is, it will return a series of directives, variables, and operators, with line numbers.

The listing will not necessarily match *exactly* the way in which a statement was entered in the first place, but will be *syntactically* the same. For instance, if you entered 510 A\$="Example" and then used the **LIST** directive, ProvideX would return 0510 LET A\$="Example".

If you use the **LIST** directive with no file number or options, the output will be displayed on your screen. You can direct the output of the **LIST** directive to any serial or indexed file or to a device (e.g., a printer or terminal). For more information, see *File Handling* in the *ProvideX User's Guide*.



If the output of a list command goes to your terminal, the listing stops to allow you to read the program page by page. To continue the listing, simply hit the **Enter** key. To stop the listing, use the **F4** key. All other input is discarded.

If the listing of a statement exceeds the maximum allowed length of a record for a given file (e.g., 80 for terminals), the statement will be broken into multiple segments (records/lines). The first  $n$  characters (where  $n$  is the maximum segment length) will be contained in the first segment with each subsequent  $n$  characters up to the end of the statement sent in subsequent segments. Each continuation segment will be prefixed by the line number followed by a colon ":" and a space.



**Note:** If a *semicolon* (;) is used as the first character of a line, ProvideX will hide the line from listings making it appear as if it did not exist. The line will execute correctly, but it cannot be viewed via the **LIST** directive.

### Coloured Syntax Displays

If the '**CS**' system parameter is *on*, your listing will be displayed with a different colour for each syntax element. For more information, refer to the '**CS** System Parameter, p.660' and the '**\*H**' Mnemonic, p.613. There is a command line utility called COLOUR (or COLOR) that can be used to display or alter the current settings. Type COLOUR or COLOR at a ProvideX prompt to display online help for this utility.

### Highlighted Search Strings

If search has been implemented for the currently-loaded program, and the '**LM**' parameter is *on*, the system will automatically highlight the search string when the program is listed. For a description of the search utility syntax, refer to [Punctuation/Syntax , p.25](#).

## Examples

Given a file with twenty character records,

```
0020 PRINT "The quick brown fox licked the dog"
```

*becomes*

```
0020 PRINT "The quic
0020: k brown fox li
0020: cked the dog"
```

| LIST Directive | Result:                                                                                  |
|----------------|------------------------------------------------------------------------------------------|
| LIST 30        | Lists line 0030                                                                          |
| / 30           | Also lists line 0030                                                                     |
| LIST 30,100    | Lists lines 0030 to 0100                                                                 |
| LIST 30,       | Lists lines from 0030 to the end of the program                                          |
| LIST ,WRAP_UP  | Lists lines up to and including the statement with label WRAP_UP                         |
| LIST           | Lists entire program                                                                     |
| LIST (1)       | Lists entire program to logical file #(1), which can be a file/device (e.g., a printer). |

# LIST\_BOX Directive

## Control List Box


### Formats

1. *Define/Create*: **LIST\_BOX** *ctl\_id*,@(col,ln,wth,ht)[,ctrlopt]  
Includes **Formatted**, **List View**, **Report View**, and **Tree View** list box formats.
2. *Remove*: **LIST\_BOX REMOVE** *ctl\_id*[,ERR=stmtref]
3. *Disable/Enable*: **LIST\_BOX** {DISABLE | ENABLE} *ctl\_id*[,ERR=stmtref]
4. *Hide/Show*: **LIST\_BOX** {HIDE | SHOW} *ctl\_id*[,ERR=stmtref]
5. *Force Focus*: **LIST\_BOX GOTO** *ctl\_id*[,ERR=stmtref]
6. *Signal on Focus*: **LIST\_BOX SET\_FOCUS** *ctl\_id*, *ctl\_val*[,ERR=stmtref]
7. *Load via Delimited String*: **LIST\_BOX LOAD** *ctl\_id*,dlm\_list\$,[ERR=stmtref]
8. *Load Via Array*: **LIST\_BOX LOAD** *ctl\_id*,array\_name\${ALL}[,ERR=stmtref]  
*Note*: The curly braces enclosing {ALL} are part of the syntax.
9. *Load/Delete Index Element*: **LIST\_BOX LOAD** *ctl\_id*,index,{element\$ | \*}[,ERR=stmtref]
10. *Find Element*: **LIST\_BOX FIND** *ctl\_id*,index,var\$[,ERR=stmtref]
11. *Read Current Selection*: **LIST\_BOX READ** *ctl\_id*,var\$[,mode\$][,ERR=stmtref]
12. *Read Current Index*: **LIST\_BOX READ** *ctl\_id*,var[,mode\$][,ERR=stmtref]
13. *Write Current Selection*: **LIST\_BOX WRITE** *ctl\_id*,element\$[,ERR=stmtref]
14. *Write Current Index*: **LIST\_BOX WRITE** *ctl\_id*,index[,ERR=stmtref]
15. *Clear Current Selection*: **LIST\_BOX WRITE** *ctl\_id*, ""[,ERR=stmtref]
16. *Report All Changes*: **LIST\_BOX AUTO** *ctl\_id*[,ERR=stmtref]

### Where:

**@(col,ln,wth,ht)** Position and size of the list box region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.

**array\_name\$** Name of array to load into list box. String variable followed by **{ALL}**.

**ctl\_id** Unique logical identifier for the list box (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels C<sup>TRL</sup>=4 for the  key) or **Negative CTL Definitions, p.817**. Use this value with the apostrophe operator to access various **Dynamic Properties**.

**ctl\_val** CTL value to generate when the list box gains focus.

**ctrlopt** Control options. Supported options for **LIST\_BOX** include:  
**ERR=stmtref** Error transfer  
**FMT=def\$** Define more elaborate **List Box Styles, p.182**.  
**FNT="font,size[,attr]"** Font name, size, optional properties  
Refer to the **'FONT' Mnemonic, p.609** for more details.  
**KEY=char\$** Hot key  
**MNU=ctl** CTL value associated with right-click menu event.  
**MSG=text\$** Message line

|                   |                                                                                                                                                                                                                                                       |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                   | <b>OPT=char\$</b> (See <b>LIST_BOX OPT= Settings</b> )                                                                                                                                                                                                |
|                   | <b>OWN=name\$</b> Name assigned for automated testing of this control.                                                                                                                                                                                |
|                   | <b>SEP=char\$</b> Delimiter character. Hex or ASCII string value.                                                                                                                                                                                     |
|                   | <b>TBL=char\$</b> Single character translation                                                                                                                                                                                                        |
|                   | <b>TIP=text\$</b> Mouse pointer message.                                                                                                                                                                                                              |
|                   | To change the colour, refer to the ' <b>TC= System Parameter, p.689</b> .'                                                                                                                                                                            |
| <b>dln_list\$</b> | Delimited list of elements to load. String expressions.                                                                                                                                                                                               |
| <b>element\$</b>  | Single element to load. Maximum string size 8kb. Use the <i>asterisk</i> * instead, to <i>delete</i> an element. For instance, <code>LIST_BOX LOAD 86 , 4 , *</code> removes element 4 from <b>LIST_BOX</b> 86.                                       |
| <b>index</b>      | Position of the element in the list box. Numeric expression. Integers: the index of the 1 <sup>st</sup> element is 1.                                                                                                                                 |
| <b>mode\$</b>     | String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to activate the list box ( <code>\$01\$</code> for <b>MOUSE-CLICK</b> or <code>\$0D\$</code> for <b>Enter</b> ). |
| <b>stmtref</b>    | Program line number or label to transfer control to.                                                                                                                                                                                                  |
| <b>var[\$]</b>    | Receives the value of the selected element (string variable) or index (numeric variable).                                                                                                                                                             |

### LIST\_BOX OPT= Settings

Available attribute/behaviour settings are listed below. Single characters may be combined. Invalid settings are ignored. Some settings are also used to define specific list view control types (`OPT="r"` for *Report Style* or `OPT="l"` for *List Style*), and tree view controls (`OPT="e"`).

"!" *Exclamation Mark. (Tree View only)*. Data has bitmaps or icons; e.g.,

```
0020 LIST_BOX 100,@(10,10,10,10),OPT="e,!"
```

If the `OPT=` setting includes "!" (as above), you can use **LIST\_BOX LOAD** statements to define bitmaps or icons for individual elements of a tree view. ProvideX displays the bitmap or icon to the left of the related element.

Include the image filename in **LIST\_BOX WRITE** statements to select (highlight) a tree view element with a bitmap or icon; e.g.,

```
0030 LIST_BOX LOAD 100,"{Cat.bmp}Cat/{Dog.bmp}Dog/{Hog.ico}Pig/"
```

or

```
0040 LIST_BOX WRITE 100,"{Cat.bmp}Cat"
```

Bitmaps or icons you define for individual elements using **LIST\_BOX LOAD** statements will *override* any default bitmaps or icons (defined using **FMT=**).

"#" *Pound Sign*. Users can select more than one entry from the list box. (This option is not supported for tree view.) If `OPT="#"`, and items were loaded in a single string, then when you read/write the element(s) highlighted in the list box, the item(s) will be returned in the variable using either the delimiter from the **LIST\_BOX LOAD** statement or, as default delimiter, the **SEP** character.

**Example:**

```

0010 PRINT 'CS'; BEGIN; LIST
0020 LIST_BOX 100,@(2,16,12,6),OPT="#"
0030 LIST_BOX LOAD 100,"Cat/Dog/Pig/"; WAIT 1
0040 ! If you read the ITEMS$ now, the returned value is null:
0050 LIST_BOX READ 100,ITEMS$;WAIT 1; PRINT ITEMS$+"Before Write"
0060 LIST_BOX WRITE 100,"Cat"; WAIT 1 ! Select "Cat"
0070 LIST_BOX READ 100,ITEMS$; PRINT ITEMS$
0080 LIST_BOX LOAD 100,2,*; WAIT 1 ! Removes "Dog"
0090 LIST_BOX WRITE 100,2 ! Select second item
0100 LIST_BOX READ 100,ITEMS$; WAIT 1; PRINT ITEMS$
0110 ESCAPE
-:run
Before Write
Cat/
Cat/Pig/

```

"|" *Pipe (Tree View only)*. Show connecting lines.

```
LIST_BOX 100,@(10,10,10,10),OPT="e,|"
```

If OPT="|" , the tree view displays connecting lines between various nodes of the tree.

"~" *Tilde. Standard and formatted list boxes only*. No height adjustment.

Normally, a list box displays an integral number of lines. If OPT="~" , the list box will *not* be forced to show only complete lines. ProvideX will truncate the last line horizontally (i.e., displaying a partial line to ensure that the height of the list box matches the size you defined).

"A" Auto signal is *on*. ProvideX returns a signal on all changes in the list box.

"b" Suppress column heading or expansion and collapse (**Report View** and **Tree View only**). In a report view, "b" suppresses the column header; e.g.,

```
LIST_BOX 100,@(10,10,10,10),OPT="r,b"
```

When you use "b" for a tree view, the "+" and "-" buttons (for the expansion and collapse of the tree) are suppressed; e.g.,

```
LIST_BOX 100,@(10,10,10,10),OPT="e,b"
```

"B" List box has no border or frame.

"d" List box is *permanently* disabled. (Cannot be enabled.)

"D" *or* "1". List box is initially disabled.

"e" *Define Tree View*. By default, a tree view shows the data as a tree with tree level "+" and "-" buttons (for the expansion and collapse of the tree). A tree view displays and maintains the data in sorted order. When you add elements to a tree view list box, ProvideX automatically creates and inserts the elements. Bitmaps or icons are optional. See **Tree View, p.192**.

- "E" *Edit Mode* - (**Tree View** only). This enables automatic editing; i.e., the user can double click on an item to change its value. The editing of items does not support the following input options:
- Format mask processing
  - Justification (right/centre)
  - Password masking
  - Input start at end (append text)
  - Force numeric or uppercase
  - Reverse input
  - Input length limitations
  - Signal on all keystrokes.
- "G" *Global*. Keep active when focus changes to a new/non-concurrent window.
- "h" List box is permanently hidden. (Cannot be shown.)
- "H" List box is initially hidden.
- "l" *Define List View*. Lowercase *L* defines a list view list box. This list box style displays a single-element list over multiple columns. (Columns wrap from the bottom of one to the top of the next column.) An optional bitmap/icon can be placed to the left of the data element. See **List View, p.189**.
- "p" Highlight partial matches (**Report View** and **Tree View** only). **WRITE** directives with strings are interpreted as selecting/highlighting items whose leading characters match the data you are writing. That is, if "ABC" is entered, then the first item starting with "ABC" will be highlighted (or all items starting with "ABC" if OPT="#" is set to allow multiple selections); e.g.,
- ```
LIST_BOX 100,@(10,10,10,10),OPT="l,p,"
or
LIST_BOX 100,@(10,10,10,10),OPT="r,p"
```
- "q" Disable sorting (**Report View** and **Tree View** only).
In list view (*report style*), that suppresses the use of the column header to sort data; e.g.,
- ```
LIST_BOX 100,@(10,10,10,10),OPT="r,q"
```
- In *tree view*, items added to the list are always added at the end of their respective trees; e.g.,
- ```
LIST_BOX 100,@(10,10,10,10),OPT="e,q"
```
- "r" *Define Report View*. Defines a report view list box. This is a formatted list box that allows optional headings, sorting, and other attributes. Each column of data has an optional {column-header} for sorting data. An optional bitmap/icon can be placed at the beginning of the row/line). See **Report View, p.189**.
- "s" *Scroll*. Control can scroll within a resizable/scrollable dialog box.
- "T" Strip trailing spaces when reading or writing data for a list box.

- "v" First Column highlight (**Report View only**). Controls the highlight style for individual list boxes and overrides print mnemonic '+V'; e.g.,
`LIST_BOX 100,@(10,10,10,10),OPT="r,v"`
 If the user clicks anywhere on a row, only the *first* column of the row will be highlighted. See also, '**+V**' & '**-V**' Mnemonics, [p.645](#).
- "V" Full Row highlight (**Report View only**). Controls the highlight style for individual list boxes and overrides print mnemonic '-V'; e.g.,
`LIST_BOX 100,@(10,10,10,10),OPT="r,V"`
 If the user clicks anywhere on a row, the *entire row* will be highlighted. See also, '**+V**' & '**-V**' Mnemonics, [p.645](#).
- "Z" Cursor changes to "resize" pointer if within 4 pixels from the edge of the control.

Description

Use the **LIST_BOX** directive to create a list box, a preset list of data elements from which the user can select items. Standard list boxes contain a single column of data with no formatting; however, this directive can be used to create a variety of list box styles.

List Box Styles

The following list box styles are defined using **OPT=** and **FMT=** settings:

- **Formatted**, [p.187](#). Displays multiple elements in different columns with alignment and width formatting, allowing colour mnemonics to be inserted into the data.
- **List View**, [p.189](#). Lists a single element over multiple columns, where data wraps from the bottom of one column to the top of the next.
- **Report View**, [p.189](#). Displays multiple elements in different columns (like a formatted list box) and allows column headings, sorting, bitmaps and other attributes.
- **Tree View**, [p.192](#). Displays data grouped into a tree-like structure which optionally may include + and - buttons to expand tree levels, dotted lines, and state indicators.



Note: A standard Windows list box can only use vertical scrollbars. However, you can use list view list boxes or grids to incorporate horizontal scrollbars into applications.

Users can select any element from a list of items you assign to the list box, but variable input is not allowed. That is, the user can only select - not enter - values. If you need a list box that allows both variable input and selection from a list, refer to **VARLIST_BOX Control List Box**, [p.360](#).

Dynamic Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. Complete lists of the properties available for manipulating **LIST_BOX**, **LIST_VIEW**, or **TREE_VIEW** objects are described in [Chapter 7. Control Object Properties](#), [p.701](#).

Format 1: Define/Create List Box

LIST_BOX *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]

Use this format to define or create a list box and give it a unique identifier in *ctl_id*. When a user selects an item from a list box, the item's associated *ctl_id* is used in generating a CTL value. Use an integer or numeric expression between -32000 and +32000 for the *ctl_id*. The *standard* list box format lists a single element of data in a single column format. Depending on the implementation, list box appearance and format type may be changed using the following control options:

FNT= sets the font for the list box. If you omit this option, ProvideX uses the system default font. Use **FNT="*" asterisk** to set the font to the standard text-mode fixed font.

OPT= expands the definition. See [LIST_BOX OPT= Settings, p.179](#).



Note: Some of the settings (including **OPT="r"**, **OPT="l"**, and **OPT="e"**) are used specifically for the creation of [List View](#), [Tree View](#) list boxes.

FMT= defines various attributes, including column/row sizing, titles and images depending on list box requirements. For more elaborate list box styles, see [Formatted, p.187](#), [List View, p.189](#), [Report View, p.189](#), or [Tree View, p.192](#).

The example below creates a standard list box that generates a CTL=100 when an item is selected from it and uses **FNT=text mode fixed font**. The list box is loaded with the items Dog, Cat, and Pig. The programmer writes Cat as the initial/highlighted selection and removes Dog from the top of the list.

```
0010 PRINT 'CS'; LIST
0020 LIST_BOX 100,@(2,14,12,6),FNT="*"
0030 LIST_BOX LOAD 100,"Dog/Cat/Pig/"; WAIT 1
0040 LIST_BOX WRITE 100,"Cat"; WAIT 1
0050 LIST_BOX LOAD 100,1,*; WAIT 1
0060 END
```



Note: The **LIST_BOX** formats described below apply to *all* list box types.

Format 2: Remove

LIST_BOX REMOVE *ctl_id*[,ERR=*stmtref*]

Use the **LIST_BOX REMOVE** format to delete a list box. (To delete an individual element, use the **LIST_BOX LOAD** directive instead.)

Format 3: Disable/Enable

LIST_BOX {DISABLE | ENABLE} *ctl_id*[,ERR=*stmtref*]

Use the **LIST_BOX DISABLE** format to gray out a list box so that it will be visible but *inaccessible* to users. To reactivate it, use **LIST_BOX ENABLE**.

Format 4: *Hide/Show*

```
LIST_BOX {HIDE | SHOW} ctl_id[,ERR=stmtref]
```

With the **LIST_BOX HIDE** format, the list box remains active, but is not displayed. It is still accessible programmatically. Use **SHOW** to restore the display and user access.

Format 5: *Force Focus*

```
LIST_BOX GOTO ctl_id[,ERR=stmtref]
```

Use the **LIST_BOX GOTO** format to reactivate and force focus to a list box, ready for the next user action.

Format 6: *Signal on Focus*

```
LIST_BOX SET_FOCUS ctl_id,ctl_val[,ERR=stmtref]
```

Use the **LIST_BOX SET_FOCUS** format to define an alternate CTL value to generate whenever focus shifts to the list box.

Formats 7, 8, and 9: *Load/Delete*

Use the **LIST_BOX LOAD** formats below to add or delete the elements in a list box. These formats set up the elements using a delimited string, an array of string elements or individually.

```
LIST_BOX LOAD ctl_id,dlm_list$[,ERR=stmtref]
```

Load Via Delimited String. When you load elements into a list box from a delimited string, the last character in the string must be a delimiter. That ending delimiter must be identical to the separator between the elements in the string; e.g.,

```
LIST_BOX LOAD 11000, "Fox/Cat/Dog/Cow/Sheep/Horse/Pig/Elephant/Ant/"
LIST_BOX LOAD 15000, "Fox"+SEP+"Cat "+SEP+"Dog"+SEP
```

Loading Data with Images

Bitmaps or icons can be included with data elements that are being loaded in a list view or tree view list box; however, the syntax for loading these images depends on the type of listbox created.

When a **List View** is created (OPT="1"), use control options FMT="{ }" and SEP="," to include images. The syntax is similar to the following:

```
0010 LIST_BOX 10,@(10,10,50,10),OPT="1",FMT="{ }",SEP=","
0030 LIST_BOX LOAD 10,"!Stop,Cat/!File,Dog/!Checkmark,Pig/"
```

For a **Report View** (OPT="r") use syntax similar to the following:

```
0010 LIST_BOX 10,@(10,10,50,10),OPT="r",FMT="{}[Col1]L10 [Col2]L10",SEP=","
0030 LIST_BOX LOAD 10,"!Stop,a,b/!Print,c,d/!Checkmark,e,f/"
```


When creating a *tree view* list box (OPT="e"), use the control option OPT="!". (see [Tree View, p.192](#)). In this case, image names are assigned to the elements using curly braces, as shown in the syntax below:

```
0010 LIST_BOX 10,@(10,10,50,10),OPT="e,!"
0030 LIST_BOX LOAD 10,"{!Stop}Cat/{!File}Dog/{!Checkmark}Pig/"
```

For more information on the options available for displaying internal/external images and the recognized image file types, see [Images and Icons, p.153](#) in the *User's Guide*.

LIST_BOX LOAD *ctl_id,array_name*{**ALL**}[**ERR=stmtref**]

Load Via Array. When an array is loaded into a list box, the curly braces in **{ALL}** must be included in the syntax.

LIST_BOX LOAD *ctl_id,index*,{*element*\$ | *}[**ERR=stmtref**]

Load/Delete Index Element. When a list box is loaded one element at a time, the index value refers to the index before the element to be inserted. (Use a value of 1 to insert an element at the start of the list. If the value of the index is 0 *zero*, the element will be appended to the end of the list.)

If you have more elements on the data list than the physical screen size of the list box can display at one time, ProvideX automatically supplies vertical scrollbars.

To remove or delete individual elements from the box, use **LIST_BOX LOAD** with an *asterisk* * instead of the element string; e.g.,

```
LIST_BOX LOAD 86,4,* ! * deletes element 4 from list_box 86
```

Format 10: *Find Element*

LIST_BOX FIND *ctl_id,index,var*[**ERR=stmtref**]

Use a string variable to get the specific element's text from a list box. By passing a specific element number, you can retrieve the text of that element into a string variable

Formats 11 and 12: *Read*

Use **LIST_BOX READ** formats to read which element in the list box has been selected. The string variable *mode*\$ returns a hex value showing how the element was selected. Possible values are described below:

\$01\$ for **MOUSE-CLICK**.

\$02\$ for **DOUBLE MOUSE-CLICK**.

\$0D\$ for **Enter**.

Once this value is read, it is reset to \$00\$.

ProvideX returns an EOM value and event when the user hits the **Insert** or **Delete** keys, **Insert** returns \$2D\$, **Delete** returns \$2E\$.

LIST_BOX READ *ctl_id,var\$[,mode\$][,ERR=stmtref]*

Read Current Selection. When you use **LIST_BOX READ** with a string variable, you can return the value of the currently selected element and the method used to make the selection (*mode\$*).

LIST_BOX READ *ctl_id,var[,mode\$][,ERR=stmtref]*

Read Current Index. If you use **LIST_BOX READ** with a numeric variable, you can return the element by index and the user's method of selection (*mode\$*) from the list box.



Note: If **LIST_BOX READ** includes an OPT="A" setting, all changes will have an effect on the EOM or *mode\$* value returned during the read.

Formats 13 and 14: Write Current Selection

Use the **LIST_BOX WRITE** formats to make the element the current selection, highlighted in the list box.

LIST_BOX WRITE *ctl_id,element\$[,ERR=stmtref]*

Write Selection. The string expression to write an element to a list box as the current selection must *exactly* match the value of one of the elements in the list box. Otherwise ProvideX returns Error #11: Record not found or Duplicate key on write.

Writing data with bitmaps or icons. Images can be included with specific data elements that are being written into a list box. The syntax for including images under **LIST_BOX WRITE** is similar to the syntax described for [Loading Data with Images, p.184](#).

LIST_BOX WRITE *ctl_id,index[,ERR=stmtref]*

Write Index. You can write the current selection to a list box by using its index.



Note: A **WRITE** to the list box using an index of 0 *zero* will reset all highlighted lines.

Format 15: Clear Current Selection

LIST_BOX WRITE *ctl_id,"" [,ERR=stmtref]*

Use this format to clear the currently selected entry in a list box.



Note: This behavior can be altered by use of the '+N' & '-N' Mnemonics, p.623.

Format 16: Report All Changes

LIST_BOX AUTO *ctl_id[,ERR=stmtref]*

Use the **LIST_BOX AUTO** format to have ProvideX generate a CTL value automatically whenever the current selection is changed. Use this to track changes to the highlighted selection in a list box.

Formatted

A *formatted* list box allows you to display multiple data elements over multiple columns in a table format. This type of list box is created by adding **FMT=** settings to the **LIST_BOX** definition (see [Format 1: Define/Create List Box, p.183](#)).

FMT= Alignment Settings

The following settings allow you to define the alignment of formatted list box columns:

- "Ln" Left justify alignment code followed by *n* width in column units.
- "Rn" Right justify alignment code followed by *n* width in column units.
- "Cn" Centered alignment code followed by *n* width in column units.
- "Nn" Numeric decimal point alignment followed by *n* width in column units.
- "S" Skip inputted field (don't show)
- "Bn" Insert *n* blanks
- "/" Line break

The list of columns is a space-separated string enclosed in quotation marks. Each column is formatted with an alignment code for left, right or centre (L, R, C). The width in the format definition is the display/window width, not the number of characters in the text. Each new row is delineated by a */slash*. To hide data, use "S" to indicate that a column is to be skipped – data is present, but not displayed and the user cannot gain access to the column.



Note: The maximum length of an element in a formatted list box is 256 bytes.

Example:

```
LIST_BOX 10000,@(5,5,35,10),FMT="L15 R5 N12 C3/B5 L25"
```

This loads the contents of the columns in a formatted list box from a delimited data string (positional data for the entries in the list box). The default delimiter is the **SEP** character (e.g., \$8A\$). To change this, use the **SEP=** option. Note that the value for the field delimiter (in this case, **SEP**) and the value signalling the end of row (in this case, \$0A\$) must be different. For the previous example:

```
"1123 East Main"+SEP+"Ont"+SEP+" 123.45"+SEP+"*"+SEP+"North Bay"+SEP+$0A$
```

Position	Width	Contents	Alignment
Col 1, Line 1	15	1123 East Main	L = Left
Col 2, Line 1	5	Ont	R = Right
Col 3, Line 1	12	123.45	N = Numeric/Decimal
Col 4, Line 1	3	*	C = Centred
Col 1, Line 2	5		B = Insert Blanks, 5 = Length
Col 2, Line 2	25	North Bay	L = Left

Colour in a Formatted List Box

Formatted list boxes support colours on a per column basis. To change the colour of a data element in a formatted list box, simply prefix the data with one of the mnemonics in the following chart.

Colour	Foreground Mnemonic	Background Mnemonic
Black	'BLACK'	'_BLACK'
Red	'RED'	'_RED'
Green	'GREEN'	'_GREEN'
Yellow	'YELLOW'	'_YELLOW'
Blue	'BLUE'	'_BLUE'
Magenta	'MAGENTA'	'_MAGENTA'
Cyan	'CYAN'	'_CYAN'
White	'WHITE'	'_WHITE'
Light Gray		'SF'+'_WHITE'

Normally, background colours are dimmer than their respective foreground colours. If you want to use the background (dimmer) colour for foreground text, prefix the colour with the **'SB'** (*Set Background*) mnemonic. If you want to use foreground colour brightness for the background, prefix the colour with the **'SF'** (*Set Foreground*) mnemonic. All colours are reset to **LIST_BOX** default standards at the end of each column.



Note: This is not standard text plane colour handling. To get light gray background use **'SF'+'_WHITE'** since **'_WHITE'** yields bright white.

Formatted List Box Example

The following example creates a formatted list box using the definition elements explained earlier:

```

0010 OPEN (1)"INVOICE", (2)"CUST"
0020 LIST_BOX 10, @(10,10,60,10), FMT="L8 B2 L20 C10 N15 C3"
0030 READ (1, END=1000) INV_ID$, INV_CUST$, INV_DATE$, INV_AMT, INV_STSS$
0040 READ (2, KEY=INV_CUST$) CST_NAME$
0050 LET L$=INV_ID$ ! 1st column
0060 LET L$+=SEP+CST_NAME$ ! 2nd column
0070 LET L$+=SEP+INV_DATE$ ! 3rd column
0080 LET X$=STR(INV_AMT: "$###,##0.00-")
0090 IF INV_AMT<0 THEN LET X$='RED'+X$ ! Column is red if negative
0100 LET L$+=SEP+X$
0120 LET X$='SF'+TBL(POS(INV_STSS$="ISP"), "", '_GREEN', '_BLUE', '_RED')+INV_STSS$
0130 LET L$+=SEP+X$
0140 LIST_BOX LOAD 10,0,L$
0150 GOTO 0030
1000 CLOSE (1), (2)
1010 INPUT *; IF CTL<>4 THEN GOTO *SAME

```

List View

A *list view* list box is similar to a standard list box, but it displays the data as a continuous list over multiple columns. When loading a list view you can place an optional bitmap or icon to the left of the data element. This type of list box is created by adding `OPT="l"` (lower case L) to the **LIST_BOX** definition (see [Format 1: Define/Create List Box, p.183](#)); e.g.,

```
0200 LIST_BOX 100,@(2,14,12,6),FMT="*",OPT="l"
```

Use **FMT=** to override the default column sizing of the list view (only "Ln", "Rn", and "Cn" alignment are supported). To indicate that a bitmap/icon is to be placed to the left of the data element, include a set of **{ }** curly braces in the **FMT=** string. See [Loading Data with Images, p.184](#). This type of list box also allows use of the partial match (`OPT="p"`) option (see [LIST_BOX OPT= Settings, p.179](#)).

Report View

A *report view* list box displays multiple data elements in table form (similar to a **Formatted** list box), but can also include optional headings, sorting, and other attributes. For a more elaborate version of this list box style, use a [GRID, p.143](#). When loading a report view, you can include a bitmap/icon to be displayed at the beginning of each row/line. This type of list box is created by adding `OPT="r"` (lower case r) to the **LIST_BOX** definition (see [Format 1: Define/Create List Box, p.183](#)); e.g.,

```
LIST_BOX 10000,@(5,5,35,10),FMT="[Company]L10 [Vendor]C6  
[Amount]#R12",OPT="r"
```

Report View Format Options

Use **FMT=** to define column alignment, titles, sorting, and bitmap placement:

Column Alignment Codes for column alignment are described under **FMT= Alignment Settings, p.187**; e.g.,

```
LIST_BOX 10000,@(5,5,35,10),FMT="[Company]L10 ..."
```

indicates that column 1 is *left* justified and 10 columns wide.

Column Titles Place title text within square brackets ahead of the alignment code to indicate column titles; e.g., ... [Company]L10.

Bitmaps or Icons To indicate that a bitmap/icon is to be placed at the beginning of the row/line, include a set of **{ }** curly braces in the **FMT=** string.

When using different images, ensure that they are all the same size. If different sizes are used, ProvideX treats the size of the first bitmap/icon as the size of all images. If you use internal bitmaps, ProvideX converts the background light gray to match the background colour of the **LIST_BOX** entries. See [Loading Data with Images, p.184](#).

Date Sorting Date code combinations define date values for sorting purposes. Up to three characters can be used to show the order of the date as it appears in the data (MD, DMY, MDY, YMD ...).

ProvideX doesn't translate or format the data, but recognizes and sorts it as date values. The data must contain some sort of alpha separator (dashes or slashes, etc.) between individual values; e.g.,

```
LIST_BOX 100,@(5,5,35,10),FMT="[Sent]DMY\11"
```

The [Sent] column will support sorting of dates where the data is in DMY *day-month-year* order (15-11-99, 01\JUL\2002, etc.).

ProvideX parses the data into a maximum of three fields and treats any alpha field it encounters as a month name. Month names must match those currently in use by the **DTE()** function. If the column contains additional data beyond the date, up to 10 of these additional characters are included in a secondary sort for the column.

Numerical Data Sorting

Specialized codes define numeric data for sorting purposes:

(*Pound Sign*) indicates that the data has been formatted using the **STR()** function and contains a consistent number of decimal places as applied by a format mask. Since the data is formatted, ProvideX will recognize the character you assign in the 'DP' system parameter as the decimal point for sorting. The default is 'DP'=46 or 'DP'=ASC("."); e.g.,

```
LIST_BOX 100,@(5,5,35,10),FMT="[Amount]#R12"
```

The [Amount] column will contain formatted numeric data, right justified, with a width of 12:

N indicates *unformatted* data – may contain the default ASC(". ") decimal point followed by varying numbers of digits; e.g.,

```
LIST_BOX 100,@(5,5,35,10),FMT="[Amount]NC6"
```

The [Amount] column will contain unformatted numeric data, centred, with a width of 10.

OPT= settings can also be used to refine the definition of a report view list box: "b" (suppress column header buttons), "p" (highlight partial matches) "q" (disable sorting), "v" (first column highlight), "V" (full row highlight). For complete descriptions, see **LIST_BOX OPT= Settings, p.179**

Load report view data into the columns from SEP-delimited data strings. For more information, see **Load via Delimited String, p. 184**. In the following example, ProvideX automatically supplies a horizontal scrollbar, where the total width of the data columns is greater than the list box display width:

```
0100 LET X$="[Custno]L10 [Name]L20 [Address]L20 [Category]C10 [Amount]R10"
0110 LIST_BOX 10000,@(10,10,50,10),OPT="r",FMT=X$,FNT="Arial,1"
0120 PRINT "Scroll to see columns"; WAIT 5
0130 LIST_BOX REMOVE 10000; END
```

The following chart describes **FMT=** settings for line 0100 in the previous example:

Position	Header Title	Width	Alignment
<i>Col. 1, Line 1</i>	Custno	10	L = Left
<i>Col. 2, Line 1</i>	Name	20	L = Left
<i>Col. 3, Line 1</i>	Address	20	L = Left
<i>Col. 4, Line 1</i>	Category	10	C = Centred
<i>Col. 5, Line 1</i>	Amount	10	R = Right

Report View Colours

Colour mnemonics can precede the data in the column or it can be intermixed with the data itself so more than one foreground/background colour can be set up per column. If a background colour mnemonic precedes all the data in a column, then the entire background of the column will be set to that colour. See [Colour in a Formatted List Box, p.188](#).

Row Highlighting

Report views support first column highlight (no matter where the user clicks on a row) and full-line highlight (where the entire row is highlighted). Set highlighting for individual list boxes using **OPT=** (either "v" or "V") when creating the list box. You can control the highlighting style system-wide by setting the print mnemonics '-V' and '+V'. For more information, refer to [Format 1: Define/Create List Box, p.183](#) and ['+V' & '-V' Mnemonics, p.645](#).

Example Report View

```

0010 OPEN (1)". "
0020 SET_PARAM 'SD'
0030 LIST_BOX 10,@(5,5,25,10),OPT="r",FMT="[Name]L15 {} [Size]R7"
0040 LOOP:
0050 READ (1,END=0150)F$
0060 IF MID(F$,-1)=DLM
0060:   THEN LET B$="!File",SZ$="<dir>";
0060:   GOTO LOADIT
0070 LET B$=""
0080 OPEN INPUT (2,ISZ=-1)F$
0090 LET X$=FIN(2)
0100 CLOSE (2)
0110 LET SZ$=STR(DEC(X$(1,4)))
0120 LOADIT:
0130 LIST_BOX LOAD 10,0,F$+SEP+B$+SEP+SZ$+SEP
0140 GOTO LOOP
0150 CLOSE (1)
0160 ESCAPE

```

Tree View

A *tree view* control object can be created by setting `OPT="e"` in the definition of a list box. (See [Format 1: Define/Create List Box, p.183.](#)) Tree views provide a hierarchical view of the data, using a collapsible tree structure to represent a list box. As items are loaded into the tree view, ProvideX automatically parses the entries based on your given delimiter and creates all intervening tree levels required. For instance, if you use a slash as your delimiter and load the single entry `aaa/bbb/ccc`, ProvideX will generate three entries in the tree view:

```
aaa
aaa/bbb
aaa/bbb/ccc
```

If you then load `aaa/bbb/ddd`, ProvideX only creates one new entry for `aaa/bbb/ddd` (since `aaa` and `aaa/bbb` already exist). If you add `aaa/xxx/iiid`, ProvideX creates two new entries: one for `aaa/xxx` and one for `aaa/xxx/iii`.

When an item from a branch is selected, the list box **READ** returns the item, including its parent branches. In the example above, selecting `ccc` would return `"aaa/bbb/ccc/"`.



Note: The unique logical identifier (*ctl_id*) assigned during the creation of a tree view list box can be used with the apostrophe operator to dynamically read and alter a wide variety of control attributes (properties). Available properties are described in [Chapter 7. Control Object Properties, p.707.](#)

As mentioned earlier, `OPT="e"` establishes a tree view definition. Several other **OPT=** settings can be used to refine the definition: `"!"` (bitmaps or icons), `"|"` (show connecting lines), `"b"` (suppress expansion/collapse buttons); `"E"` (enable automatic editing), `"q"` (disable sorting). For complete descriptions, see [LIST_BOX OPT= Settings, p.179.](#)

Tree View Format Options

Use the optional **FMT=** setting to define *default* images to be displayed in the tree. To add images, enclose the image name in curly braces. For more information on the options available for displaying internal/external images and the recognized image file types, see [Images and Icons, p.153](#) in the *User's Guide*.

Filenames are mandatory in **FMT="{images\$}"** clauses. Place them inside curly braces in a pipe-separated list; e.g.,

```
LIST_BOX A,@(x,y,10,10),OPT="e",FMT="{Cat.ico|Dog.bmp|Pig.bmp}"
```

Null values in **FMT="{images\$}"** are not allowed. For instance, `FMT="{||Pig.bmp}"` generates an Error #23: Missing/Invalid variable.)

Ensure that all bitmaps or icons in the same **LIST_BOX** control are the same size (mandatory). If you include different sizes, ProvideX treats the size of the first bitmap/icon as the size of all images for a given list box.

If you use internal bitmaps, ProvideX converts the background light gray to match the background colour of the **LIST_BOX** entries. You can define up to six *default* bitmaps or icons in tree views.

The order of the images determines when they are used:

1. Default overall bitmap or icon: always used with any listed entries that do not have subordinates.
2. Default bitmap or icon for items with subordinates.
3. Default bitmap or icon for items with subordinates if the tree level is expanded (i.e., shown) in tree view.
4. Bitmap or icon for entries that do not have any subordinates when the item is selected.
5. Bitmap or icon for entries that have subordinates when selected.
6. Bitmap or icon for entries that have subordinates when selected and level is expanded.



Note: If images are set up for *individual elements* in tree view **LIST_BOX LOAD** and **WRITE** statements, these will *override* the default **FMT=** images for the individual element. For more information, see [Loading Data with Images, p. 184](#).

Example Tree View

```
0010 LIST_BOX 10,@(5,5,25,10),OPT="e|",
0010:FMT="{!diskette|!File|!File_open}",SEP=DLM
0020 LET F=1,D$="."
0030 NXTDIR:
0040 SET_PARAM 'SD'=1
0050 OPEN (F)D$
0060 NXTFILE:
0060:READ (F,END=ENDDIR)F$
0070 IF F$(1,1)="."
0070:THEN GOTO NXTFILE
0080 IF MID(F$,-1)<>DLM
0080:THEN LIST_BOX LOAD 10,0,PTH(F)+DLM+F$;
0080:GOTO NXTFILE
0090 LET D$=PTH(F)+DLM+F$
0100 F++
0110 GOTO NXTDIR
0120 ENDDIR:
0120:CLOSE (F)
0130 F--
0140 IF F>0
0140:THEN GOTO NXTFILE
0150 ESCAPE
```



Tip: Use **SEP=DLM** when reading directories to have ProvideX append the operating system delimiter to subdirectory names.

LOAD Directive

Read Program into Memory

Format **LOAD** [*prog_name*\$_]

Where:

prog_name\$ Name of the program to load/read. String expression. To load a program from a program library, use [\[LIB\]](#), [p.781](#).

Description Use the **LOAD** directive to read a program into memory (i.e., to open or retrieve a program) for execution, listing, or modification. The program you load into memory replaces the current program, if any.

On a **LOAD**, ProvideX resets the **FOR/NEXT**, **GOSUB/RETURN**, and **WHILE/WEND** stack. Any addresses set in **SETERR** or **SETESC** directives are cleared and the current **PRECISION** is reset to two. The current **DATA** pointer is also reset to the start of the program. No user data areas or files are affected by the **LOAD** directive.

You can only use the **LOAD** directive in Command mode or with the **EXECUTE** directive. Misuse of the **LOAD** directive in Execution mode (i.e., without the **EXECUTE** directive) generates Error #45: Referenced statement invalid.

If you omit the program filename, ProvideX loads the last file for which there was a **LOAD**, **RUN**, or **SAVE**. If there is no prior file in the stack, ProvideX returns Error #10: Illegal pathname specified.

See Also [RUN Transfer and Execute a Program, p.294](#)
[SAVE Write Program to File, p.295](#)
[Saving, Loading, and Executing a Program, User's Guide](#)



Note: ProvideX accepts certain typographical errors. For instance, it accepts **LAOD** as a substitute for **LOAD**.

Examples

```
LOAD "MYPROG" ! Loads "MYPROG"
LOAD ! Loads last program specified, if any (in this case, "MYPROG")

->DELETE
->LOAD
Error #10: Illegal pathname specified
```

LOAD CLASS Directive Pre-Load Class Definition

Format `LOAD CLASS class$ [FROM "filename$"]`

Where:

class\$ Name of the class to be pre-loaded. String expression.

filename Optional. Filename (.pvc) if different than *class\$*.

Description The **LOAD CLASS** directive is used in *Object Oriented Programming* to pre-load a class definition into memory from a .pvc file. A `LOAD CLASS "aaa"` without any *filename\$* specified will auto-load from `aaa.pvc`. The first code in the .pvc file must be a **DEF CLASS** directive, using the same class name (*class\$*) as is specified in the **LOAD CLASS** directive.

Normally an OOP class definition is loaded into memory the first time an object of that class is instantiated, and that class definition remains in memory and is used for all subsequent instances of the same class.

See Also

[DEF CLASS Define Object Class, p.65](#)

[DROP CLASS Delete Class Definition, p.102](#)

[DROP OBJECT Delete Object, p.104](#)

[RENAME CLASS Change Name of Class, p.283](#)

[STATIC Add Local Properties at Runtime, p.329](#)

[NEW\(\) Function, p.489](#)

[REF\(\) Function, p.512](#)

[Data Integration, User's Guide](#)



Note: ProvideX accepts certain typographical errors. For instance, it accepts **LAOD** as a substitute for **LOAD**.

LOAD DATA Directive

Load Program Constants

Format **LOAD DATA** *filename*\$ [,**ERR=***stmtref*]

Where:

filename Name of Variable Definition file in which to store constant.

stmtref Program line number or label to transfer control to.

Description This directive loads into memory the contents of a Variable Definition file (created via the **SAVE DATA** directive). These variables are read-only, and any attempt to change them will result in an Error #61: Authorization failure. Global variables are not supported.

See Also **SAVE DATA Save Program Constants, p.297.**

Example

```
COMPANY$="ABC Company"
DIVISION$="Laundry Division"
COMPANY_CODE=1
SAVE DATA "CO_DATA",COMPANY$,DIVISION$,COMPANY_CODE

START

LOAD DATA "CO_DATA"

DUMP
! ERR=0, CTL=0, RET=2
! Level=1
! PGN="<Unsaved>"
! Loaded data...CO_DATA (C:\Documents and Settings\Default
    User\Application Data\CO_DATA)
COMPANY$="ABC Company"
DIVISION$="Laundry Division"
COMPANY_CODE=1

COMPANY_CODE=2
Error #61: Authorization failure
```

LOCAL Directive

Designation of Local Data

Format

1. *Assign Local Variable*: **LOCAL** *varlist*
2. *Assign Local Arrays*: **LOCAL DIM** *array_name*[\$]
3. *Define Local Properties (OOP)*: **LOCAL** *prop1*[**OBJECT**], *prop2* [**OBJECT**],...

Where:

array_name[\$] Numeric or string variable to be dimensioned as an array.

OBJECT Optional keyword identifies that the property contains an object identifier to another object.

prop1,
prop2 ... Property names – treated like any other variable in the system. (In *Object Oriented Programming*, data elements are called *properties*.)

varlist Variables to be used temporarily in the execution of a subroutine, subprogram, **FOR/NEXT**, **SWITCH/END SWITCH**, **WHILE/WEND**, **REPEAT/UNTIL**, or user-defined function.

Description

The **LOCAL** directive is used to reassign variable names temporarily within a called procedure, without affecting the original contents (if any). In *Object Oriented Programming* (OOP), the **LOCAL** directive is similar to the **PROPERTY** directive, but is used to declare data that is only visible to processing logic within the object itself.

Format 1: Assign Local Variables

LOCAL *varlist*

Use this format to reassign variable names temporarily, without affecting the original contents. If the variable name supplied in the **LOCAL** directive is in current use, ProvideX will preserve its value/contents. Once the current **FOR/GOSUB** stack has been cleared or the program is exited, ProvideX restores variables that were designated local to their original values. (Local variables are only active until the current stack is cleared.)

The **LOCAL** directive can take an assignment during declaration; however, direct assignment does not work for arrays that are declared as local.

Example:

In the example below, X is designated as local for both subroutines (CHK_IT and LOOP), so the value of X is restored to its original value "1234" after the GOSUB stack is cleared for each subroutine. Since X\$ was not designated as **LOCAL** in the LOOP subroutine, its value has changed from "START TEST, X=" to "LOOP" after the GOSUB stack for the LOOP has been cleared.

```

1030 LET X=1234,X$="START TEST, X="; PRINT X$,X
1040 GOSUB CHK_IT
1050 PRINT "TEST DONE"
1060 GOSUB LOOP
1070 PRINT X$,X," DONE"; STOP

```

```

6000 CHK_IT:
6010 LOCAL X$,X; LET X$="CHECK X:"; PRINT X$,X
6020 LET X=X+10
6030 PRINT X$,X
6040 RETURN
7000 LOOP:
7010 PRINT "START LOOP, X=",X
7020 LET X$="LOOP" ! Not designated as LOCAL
7030 FOR LOCAL X=1 TO 4
7040 PRINT X$,X
7050 NEXT X
7060 RETURN
-:run
START TEST, X= 1234
CHECK X: 0
CHECK X: 10
TEST DONE
START LOOP, X= 1234
LOOP 1
LOOP 2
LOOP 3
LOOP 4
LOOP 1234 DONE

```

Variables in function definitions can be designated as local to prevent changes in program variables:

```
0010 DEF FNXY(LOCAL X,LOCAL Y,LOCAL Z)=X+Y+Z
```

Format 2: Assign Local Arrays

LOCAL DIM *array_name*[\$]

Use **LOCAL** with **DIM** to define an array name temporarily within a called procedure. Once defined, subsequent uses of **DIM** to assign dimensions to the array will be assumed to be local.

In *Object Oriented Programming* (OOP), array names are assigned local in the same way as variables. The **DIM** directive is not required; e.g., **LOCAL array_name**[\$].

Format 3: Define Local Properties in Object Oriented Programming

LOCAL prop1[**OBJECT**], **prop2** [**OBJECT**]....

In *Object Oriented Programming*, the **LOCAL** directive can be used to define the properties for an object class that are not exposed to external applications. They can be accessed during the execution of program logic within the object class itself.

This format of the **LOCAL** directive is similar to the **PROPERTY** directive: variable declarations may include dimensioned arrays and object references. **LOCAL** properties are typically used for:

- File handles (if the object is maintained on a file).
- Security information.
- Flags and status information used by the programming logic.

If the local variable contains an object identifier to another object, specify the keyword **OBJECT** after the variable *name*. When the object is deleted, ProvideX will use the **REF()** function against the object identifier to remove it (as long as it has no other references); e.g.,

```
DEF CLASS "Customer"  
LOCAL File OBJECT
```

When you delete an object whose class is `Customer`, then the system reduces the reference count of the object whose identifier is in `File` and, if it is no longer being referenced, deletes it as well.

See Also

[DEF CLASS Define Object Class, p.65](#)
[DROP CLASS Delete Class Definition, p.102](#)
[DROP OBJECT Delete Object, p.104](#)
[FUNCTION Declare Object Method, p.137](#)
[LIKE Inherit Properties, p.174](#)
[LOAD CLASS Pre-Load Class Definition, p.195](#)
[PROGRAM Create/Assign Program File, p.259](#)
[PROPERTY Declare Object Properties, p.261](#)
[RENAME CLASS Change Name of Class, p.283](#)
[STATIC Add Local Properties at Runtime, p.329](#)
[NEW\(\) Function, p.489](#)
[REF\(\) Function, p.512](#)
[Data Integration, User's Guide.](#)

LOCK Directive

Reserve File for Exclusive Use

Format **LOCK** (*chan*[,**ERR**=*stmtref*])

Where:

chan Channel or logical file number of the file to be locked.

stmtref Program line number or label to transfer control to.

Description Use the **LOCK** directive to reserve a given file for exclusive processing by the user/program. Once a file is locked, no other user or program can gain access to it. However, if another user already has the given file open, the **OPEN** and **LOCK** directives fail and ProvideX returns Error #0: Record/file busy.

See Also [UNLOCK Remove Exclusive Use from File, p.349](#)

Examples

```
0010 OPEN (30,ERR=0100)"GLFILE"  
0020 LOCK (30,ERR=0120)  
...  
0100 PRINT "Cannot open GLFILE"  
0110 STOP  
0120 PRINT "GL still in use—try later"  
0130 STOP
```


LONG_FORM Directive

Use Long Variable Names

Format **LONG_FORM**

Description Use the **LONG_FORM** directive to have the ProvideX compiler's input-parsing routine accept long variable names (default mode). The complementary directive is **SHORT_FORM**, which allows only short variable names.

You can write programs in either **SHORT_FORM** or **LONG_FORM** or a combination of both. You can also run programs in either mode.

See Also ['LF' System Parameter, p.672](#),
[SHORT_FORM Use Short Variable Names, p.325](#).

Examples

```
-:LONG_FORM
-:LONG_NAME$="OK"
-:SHORT_FORM
-:LONG_NAME$="NOT OK IN SHORT_FORM"
Error #20: Syntax error ...ONGNAME$="L... (Long variable name not
      accepted)
-:L$="OKAY IN SHORT_FORM"
```

MENU_BAR Directive

Control Menu Bar

Formats

1. *Define/Create*: `MENU_BAR ctl_id,menu_def$[,ERR=stmtref]`
2. *Remove*: `MENU_BAR REMOVE[,ERR=stmtref]`
3. *Disable/Enable Item*: `MENU_BAR {DISABLE | ENABLE} element[$][,ERR=stmtref]`
4. *Force Focus*: `MENU_BAR GOTO[,ERR=stmtref]`
5. *'Check'/'Uncheck' Item*: `MENU_BAR {ON | OFF} element[$][,ERR=stmtref]`
6. *Read Selected Item*: `MENU_BAR READ var$[,ERR=stmtref]`
7. *Clear Menu Bar*: `MENU_BAR CLEAR[,ERR=stmtref]`
8. *Restore Default Help*: `MENU_BAR RESET[,ERR=stmtref]`

Where:

<i>ctl_id</i>	Unique logical identifier for the menu bar. Use an integer between -32000 to +32000. Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the F4 key) or special negative CTL values set by the system. See Negative CTL Definitions, p.817 .
<i>element[\$]</i>	Individual menu selection. Expression consisting of shortcut letters or assigned item CTL values for accessing the selection.
<i>menu_def\$</i>	Menu structure and elements. String expressions. Maximum 2047 elements. A bitmap/icon can be included for each element in the menu. <i>Tip:</i> To determine how many elements you already have, count the ampersands (&).
<i>stmtref</i>	Program line number or label to transfer control to.
<i>var\$</i>	String variable to receive the last selected menu item.

Description Use the **MENU_BAR** directive to define and control the menu across the top of a window.

Format 1: Define/Create Menu Bar

`MENU_BAR ctl_id,menu_def$[,ERR=stmtref]`

This format creates a menu bar control object. If the menu bar is not properly defined, ProvideX returns an Error #87: MENUBAR definition invalid. Define the *menu_def\$* of menu groups and elements and identify each item uniquely (e.g., F and O would identify File/Open) by marking the character the user can enter in conjunction with the **Alt** key (e.g., **Alt-F**) to select the particular item. Use the following format:

- Enclose each menu group in square brackets.
- Delimit each item in a group with a comma.
- Prefix each item's selection character (hot key) with & (an ampersand).
- Prefix each sub-menu group with its item ID.

The example below shows a menu string containing File and Edit, each with a submenu (File: Open, Save, and Quit – Edit: Add and Delete). Each item has a shortcut key. Quit has both the shortcut key, Q, and a control ID value, 4.

```
MENU_BAR 100, "-[&File,&Edit],F:[&Open,&Save,&Quit=4],E:[&Add,&Delete]"
```

ProvideX includes the HELP selection on all menu bars by default. You can suppress the HELP selection by inserting a *minus sign* "-" as the first character in a menu definition string, as in the example above.

Right-justify a description of the menu entry (e.g., to tell users what function key will give them quick access to the selection). Use the tab character \$09\$ to separate the text to be right-justified; e.g.,

```
MENU_BAR 99, "[&File,&Edit...],E:[&Cut"+$09$+"Shft-DEL,&Paste"+$09$+"Ins]"
```

Assigning CTL Values

Normally, any selection from the menu bar will generate its *ctl_id*. You can also have individual menu items generate CTL values. To do this, append an *equals sign* "=" and CTL value to any item in the menu selection list (e.g., &Quit=4, in the example above).

It is best to use unique shortcut keys for selections from a given sub-menu in a group. If you have duplicates, you'll have difficulty determining which selection a user makes unless a unique CTL value is assigned to each item. If you omit a shortcut key, ProvideX assigns a value equivalent to the item's placement in the definition string.

Add a line to separate options by inserting an additional comma ",," as a place holder in the definition string (e.g., between &Open &Save).

```
MENU_BAR 120, "[&File],F:[&Open,,&Save,&Quit=4]"
```

Menu items can also be *disabled*, or displayed in bold or with a checkmark, by placing a "D", "B", or "C" after the = (equal sign) and before the assigned CTL value.; e.g., "[&One=1,&Two=BC2,&Three=D3]"

Using Images

Images can be included for each item in the menu. Enclose the image name in curly braces and place it in the menu definition just prior to the specific item text; e.g.,

```
MENU_BAR 1000, "-[&File],F:[&Open=1001,{!Stop}&Stop=1002]"
```

Use a leading exclamation point (!) to identify the image as internal, or specify the relative path and filename to access an image file that is external. The first bitmap determines the dimensions used to display menu items (up to 64x64). Transparency options can also be included. "T" indicates use of the upper left most pixel colour, and "G" means use colour RGB: 192,192,192; e.g.,

```
E:[{!Copy,t}&Copy,{!Paste,g}&Paste]
```

For more information on the options available for displaying internal/external images and the recognized image file types, see [Images and Icons, p. 153](#) in the *User's Guide*.

Two-Tone Effects

Both the **MENU_BAR** and **POPUP_MENU** directives support the use of two optional parameters for defining *background* and *left edge* colours in menus (similar to MS Office applications):

- LEFT(arg)** Background colour for the bitmap portion of the menu. Must be placed outside "[...]" menu definitions.
- FILL(arg)** Background colour for the right/text side. If placed outside "[...]" menu definitions it will serve as the default colour for all menu items. If placed within "[...]", it will be considered the colour associated with a specific menu item.

The *arg* value is defined using the same format as the **Colour Properties, p.727**. In the following example, the RGB colour 200,200,200 is used for the left edge of all entries in this menu:

```
MENU_BAR 10, "LEFT(RGB:200,200,200), [&File,&Edit,&Help], F:[...]"
POPUP_MENU "LEFT(RGB:200,200,200), [&Cut,&Paste,&Delete]", x$
```

The next example uses the RGB colour 255,255,150 as the background colour for all of the text portion of the menu.

```
MENU_BAR 10, "FILL(RGB:255,255,150), [&File,&Edit,&Help], F:[...]"
POPUP_MENU "FILL(RGB:255,255,150), [&Cut,&Paste,&Delete]", x$
```

In the following, the menu will have the "Delete" entry highlighted with a different colour background:

```
MENU_BAR 10, "[&Edit], E:[&Cut,&Paste,&Delete=FILL(RGB:255,255,192)]"
POPUP_MENU "[&Cut,&Paste,&Delete=FILL(RGB:255,255,192)]", x$
```



Note: The colours only affect vertical portions on the menu, not the area that runs horizontally across the top. Also note that system-supplied Cut, Copy, Paste, and Delete menu items will adhere to the **FILL(arg)** and **LEFT(arg)** definitions for the menu.

Format 2: Remove Menu Bar

MENU_BAR REMOVE[,ERR=*stmtref*]

This format deletes the menu bar completely from the current session. It cannot be re-displayed until it is redefined.

Format 3: Disable/Enable Item

MENU_BAR {DISABLE | ENABLE} element[\$][,ERR=*stmtref*]

Use the **MENU_BAR DISABLE** format to gray out the specified menu bar item so that it will be visible but *inaccessible* to users. **ENABLE** reactivates it. See "**Toggling**" above.

Format 4: Force Focus

MENU_BAR GOTO[,ERR=*stmtref*]

This enables and forces focus to a menu bar, ready for the next user action.

Format 5: 'Check'/'Uncheck' Item

MENU_BAR {ON | OFF} *element*[\$][,ERR=*stmtref*]

The **MENU_BAR ON** option displays a check mark in front of a specified menu bar item, making it appear that it was selected. **OFF** removes the check mark.

Toggleing CTL values (Formats 6 and 7): Menu items *element*[\$] are usually identified via shortcut keys; e.g., **MENU_BAR ON "FPS"** toggles items **F**ile, **P**rint, **S**ave. When using CTL values, and more than one menu item uses the same CTL, then all menu items using that CTL will be toggled. If toggling just one item, then use that CTL value only. If toggling more than one, then make a string of the CTL values, prefixing a pound sign (#) to each CTL value separated by commas; e.g.,

```
MENU_BAR ON 12034
MENU_BAR OFF "#12034,#12035,#12036"
```



Note: Formats 6 and 7 result in an Error #11: Record not found or Duplicate key on write if a **MENU_BAR** item cannot be found.

Format 6: Read Selected Item

MENU_BAR READ *var*[\$][,ERR=*stmtref*]

When a user selects any of the items from a menu bar, ProvideX generates the *ctl_id* you assigned to the menu bar. You can return the selected menu item code in a string variable using **MENU_BAR READ**. Based on the following example, if the user selected *Delete* from the *Edit* menu, **MENU_BAR READ** would return ED:

```
MENU_BAR 100, "-[&File,&Edit],F:[&Open,&Save,&Quit=4],E:[&Add,&Delete]"
```

Format 7: Clear Menu Bar

MENU_BAR CLEAR[,ERR=*stmtref*]

Use the **MENU_BAR CLEAR** format to remove the menu bar from the current window until a **BEGIN** or **END** is executed.



Note: Use a **MENU_BAR RESET** or **MENU_BAR CLEAR** directive before altering the menu bar.

Format 8: Restore Standard Help

MENU_BAR RESET[,ERR=*stmtref*]

This format resets the current menu bar to the default Help setting.

MERGE Directive Read/Append Lines from File

Formats

1. *Specify Source File by Channel:* **MERGE** (*chan,fileopt*)
2. *Specify Source File by Serial Filename:* **MERGE** *filename\$*

Where:

chan Channel or logical file number of the file to reference.

fileopt Supported file options (see also, [File Options, p.810](#)):

ERR=stmtref Error transfer

IND=num Record index

TBL=stmtref Record number.

filename Name of the serial file to be used as the source file for the merge.

Description

Use the **MERGE** directive to add program statements from a source file to the current (target) program. You can use a **MERGE** directive for serial or indexed files. You can also use it with Memory files. The source file must contain records that are statements in valid List format (i.e., with correct syntax, line numbers, etc.).



Warning: During the **MERGE** process, if a statement from the source file has a line number matching a line number in the target program, the statement from the source file will overwrite the corresponding statement in the target program.

You have to include line numbers in the source file, but you do not have to put them in numeric order. ProvideX reads them into the target program in ascending sequence as if they were entered in Command mode, but will place them in the correct numeric sequence, using the source file's numbering.

If a source file contains a statement without a statement number, or with an invalid statement number (outside of the allowed range), ProvideX generates Error #21: Statement number is invalid and halts the **MERGE** process.

ProvideX terminates the **MERGE** process when it encounters an End-of-File status in the source file.



Note: **MERGE** does not support code generated using the **LIST EDIT** format.

Format 1: Specify Source File by Channel

MERGE (*chan,fileopt*)

This opens the source file and use its current channel / logical file number to identify it. You can use this format with a memory-resident file (i.e., ***MEMORY***).

Format 2: Specify Source File by Serial Filename

MERGE *filename*\$

You can use a serial filename to identify it as the source for the **MERGE** (instead of opening it and referring to the channel).

Example

The following is a sample terminal session using the **MERGE** directive to combine two programs:

```

0->LOAD "TIME.PRT"
->LIST
1000 REM Time output routine
1010 T$=STR(INT(TIM)*100+FPT(TIM)*60:"00:00")
1020 RETURN
->SERIAL "WORKFILE"
->OPEN (1)"WORKFILE"; LOCK(1)
->LIST (1)
->CLOSE (1) ! WORKFILE has list of program
->LOAD "PASS.CTRL"
->LIST
0010 GOSUB 1000; PRINT "1st pass: ",T$
0020 CALL "PASS1"
0030 GOSUB 1000; PRINT "2nd pass: ",T$
0040 CALL "PASS2"
0050 GOSUB 1000; PRINT "End both: ",T$
0060 STOP
->OPEN (1)"WORKFILE"
->MERGE (1)
->CLOSE (1)
->LIST
0010 GOSUB 1000; PRINT "1st pass: ",T$
0020 CALL "PASS1"
0030 GOSUB 1000; PRINT "2nd pass: ",T$
0040 CALL "PASS2"
0050 GOSUB 1000; PRINT "End both: ",T$
0060 STOP
1000 REM Time output routine
1010 T$=STR(INT(TIM)*100+FPT(TIM)*60:"00:00")
1020 RETURN

```

See Also

[INDEXED Create Indexed File, p.159](#),
[*MEMORY* Create & Use Memory File, p.741](#),
[SERIAL Create a Sequential File, p.302](#)



MESSAGE_LIB Directive *Establish Message Library*

- Formats
1. *Designate Message Library*: **MESSAGE_LIB** *filename*[\$[,NBF=*num*]][,ERR=*stmtref*]
 2. *Add Library to Top of List*: **MESSAGE_LIB ADD** *filename*[\$[,NBF=*num*]][,ERR=*stmtref*]
 3. *Remove Library from List*: **MESSAGE_LIB DROP** *filename*[\$[,ERR=*stmtref*]
 4. *Remove Top Library from List*: **MESSAGE_LIB POP** [,ERR=*stmtref*]

Where:

filename Name of the variable length Direct/Keyed file to be used as a Message Library. String expression.

num Number of buffers to allocate.

stmtref Program line number or label to transfer control to.

Description Use the **MESSAGE_LIB** directive to designate files as containing messages to be returned by the **MSG()** function. A message library must be a variable length Direct / Keyed file with the message identifier as the key. The record contains the message. As of Version 4.12, there is an alternate key to the `*msglib.en` (and any newly created **MESSAGE_LIB** files) on the first 50 characters of the message field.

See Also [MSG\(\) Function, p.484](#)
[KEYED Create Single/Multi-Keyed File, p.166](#)
[DIRECT Create File with Keyed Access, p.89](#)

Format 1: *Designate Message Library*

MESSAGE_LIB *filename*[\$[,NBF=*num*]][,ERR=*stmtref*]

Use this format to designate and add *filename*(s) to the **MESSAGE_LIB** list.

In earlier versions of ProvideX, only one message library could be active at a time – when using a **MESSAGE_LIB** directive. ProvideX closed any other currently active message library and designated your given file as the new one. Currently, you can now have more than one message library active at a time. With use of the **MSG()** function, the message libraries are searched in order until a match is found.

An error is generated if the file cannot be properly opened. Use **NBF=** to specify the number of buffers to allocate. The *filename*\$ expression can contain one or more **MESSAGE_LIB** filenames, each **SEP**-separated, so that you can reset the entire list in one command.

Example 1:

```
KEYED "MESSAGE.LIB",10,0,-256
OPEN (1)"MESSAGE.LIB"
WRITE RECORD (1,KEY="NOCUST")"Sorry but Customer %1 is not valid"
CLOSE (1)
MESSAGE_LIB "MESSAGE.LIB"
PRINT MSG("NOCUST","0001")
Sorry but Customer 0001 is not valid
```

Example 2:

```
0010 MESSAGE_LIB "*MSGLIB."+ENV("LANG"),ERR=0020;GOTO 0100
0020 MESSAGE_LIB "*MSGLIB.EN"
0100 REM...
0110 ! ...
1000 READ (CST_FN,KEY=K$,DOM=1900)...
1010 ! ...
1900 PRINT MSG("REC_MISS",K$)
```

Format 2: Add Library to Top of List

MESSAGE_LIB ADD *filename\$* [,NBF=*num*] [,ERR=*stmtref*]

Use this format to add a *filename\$* to the top of the list of message library to search. An error is generated if the **MESSAGE_LIB** file cannot be properly opened. Use the **NBF=** option to specify the number of buffers to allocate.

Format 3: Remove Library from List

MESSAGE_LIB DROP *filename\$* [,ERR=*stmtref*]

This format removes the specified *filename\$* from the list of message libraries to search. An Error #12: File does not exist (or already exists) is generated if the *filename\$* is not on the list.

Format 4: Remove Top Library from List

MESSAGE_LIB POP [,ERR=*stmtref*]

Use this format to remove the top *filename\$* from the list of message libraries to search. An Error #12: File does not exist (or already exists) is generated if *filename\$* is empty.

MNEMONIC Directive Define File Command Sequence

Format **MNEMONIC** [(chan)]mnm_name\$=esc_sequ\$

Where:

chan Channel or logical file number for which the mnemonic is defined.
mnm_name\$ Valid mnemonic name, enclosed in apostrophes. String expression.
esc_sequ\$ Escape or command sequence required by the file/device in order to process the mnemonic. String expression.

Description Use the **MNEMONIC** directive to define additional mnemonics for files and/or devices. Once you define a mnemonic for a given logical file number, it stays active until the channel is closed. When ProvideX encounters a user-defined mnemonic in a **PRINT** or **INPUT** statement, it converts the mnemonic to the character string in the escape sequence.

Under WindX

WindX supports the use of this **MNEMONIC** directive via the [WDX] tag. After opening a channel across a WindX connection, all declared mnemonics (except '*R' and '*X') are sent automatically to the WindX workstation. ProvideX considers the '*R' and '*X' mnemonics to be local to the server unless you use the [WDX] tag in declaring them. ('*R' declares an operating system command to execute on channel close, and '*X' declares a program to call on channel close.)

The following is an example of the **EXECUTE** command under WindX *prior to Version 4.20*:

```
IF WDX%<>$0$ THEN EXECUTE "[WDX]MNEMONIC(LFO)'5X'=$1B+hex$" ELSE GOTO MY_LABEL
```

For more information, see [\[WDX\] Direct Action to Client Machine, p.801](#).

See Also [Chapter 5. Mnemonics, p.577](#).

Examples To define and use a mnemonic:

```
0010 OPEN(1) "/dev/printerxx"
0020 MNEMONIC 'US'=ESC+"_" ! Define mnemonic
....
0030 PRINT(1)'US',"Title line...", ! Underscored
```

The example below illustrates how the **MNEMONIC** directive can be used for flexible text fonts on a screen in Windows. Define the font and the logical screen size in a statement. For example, to assign settings for the ProvideX standard mnemonics 'CP' and 'SP':

```
MNEMONIC(0)'CP'="Courier New,-8":120,40
MNEMONIC(0)'SP'="*":80,25
```

Then, a subsequent PRINT 'CP' directive will change the screen font to Courier New with a size of 8 points. PRINT 'SP' restores the screen font to standard print.



Note: The TEXT plane font must always be a fixed font.

WindX Examples:

```
0OPEN(chan) "[WDX]\\mach\printer_share"  
MNEMONIC(chan)'FF'=$0C$ ! automatically goes to WindX workstation  
MNEMONIC(chan)*R'="erase "+filename$ ! local to server, not workstation  
MNEMONIC(chan)*R'="[WDX]erase "+filename$ ! on workstation, not server
```

MSGBOX Directive

Display PopUp Message Box

Format **MSGBOX** *text*[\$[,*title*[\$[,*options*[\$[,*selection*[\$]]]]]

Where:

options\$ Customized message box options. Optional string expressions, separated by commas if you include a list. Supported options for **MSGBOX** buttons include:

OK	Ok only
CANCEL	Ok, Cancel
RETRYCANCEL	Retry, Cancel
ABORT	Abort, Retry, and Ignore.
YESNO	Yes, No
YESNOCANCEL	Yes, No, Cancel
1 (or 2 or 3)	Default button number

Valid icons (in graphics mode only):

STOP	Stop sign
INFO	Lowercase 'i' in a circle.
QUESTION or ?	Question mark
EXCLAMATION or !	Exclamation Mark

Miscellaneous:

BEEP	Send associated sound
TIM=num	Maximum time-out value in integer seconds. This allows message boxes to time-out and close automatically. The returned value is "TIMEOUT".
TOP or ^	Always on top (forces foreground window)

Example: **MSGBOX** "The report is completed","F.Y.I.", "TIM=3"

In the above example, the message box will self-destruct in 3 seconds..

selection\$ String variable to return the button selected by the user. Possible values are "**ABORT**", "**CANCEL**", "**IGNORE**", "**NO**", "**OK**", "**RETRY**", "**YES**".

text\$ Text to appear in the message box. String expression.

title\$ Title of the message box. String expression. (If you omit this field, the title will be "**Error**".)

Description

Use the **MSGBOX** directive to display a window with a message in the middle of the screen. The text will be split into segments (lines) based on the system settings for screen width and centring characteristics. You can define multiple lines of text by using a **SEP** character between lines. Use the options string to identify the buttons and/or icons you want to include in the message box. The user's message box button selection is returned in a string variable.

Customizing the Message Box

If the 'MX' system parameter is set, the system calls subprograms `*ext/msgbox.gui` (user-defined) or `*ext/system/msgbox.gui` (ProvideX-supplied) to process the request instead of the standard Windows message box. By default, ProvideX sets the 'MX' parameter to *On* when `*ext/msgbox.gui` is found to exist.

The `msgbox.gui` subprogram creates and displays a message box that is virtually identical to the standard Windows system message box but will use XP-style (or Vista) buttons if the '4D' mnemonic is enabled. In addition, it will use the currently-selected windows graphic font.



Note: When 'MX' is set, **MSGBOX** commands entered in console mode or executed within an **EXECUTE** command *cannot* be followed by any other command (as **MSGBOX** will be executing a **CALL** without a return address).

Several internal bitmap names for standard Windows bitmaps are available for displaying the embedded OS icons used by the normal message box API call: `!Sys_Stop`, `!Sys_Question`, `!Sys_Info`, and `!Sys_Exclamation`.

The button text OK, YES, NO, CANCEL, ABORT, RETRY, CONTINUE, and IGNORE are included in the system message library file (i.e., `*mlfile.en`) to provide support for multi-lingual systems. The message numbers are defined as follows:

MSG() Number	String
160	"OK"
161	"OK, Cancel"
162	"&Retry, Cancel"
163	"&Abort, &Retry, &Ignore"
164	"&Yes, &No"
165	"&Yes, &No, &Cancel"

Use the **DEF MSG** directive to temporarily override the message text associated with the **MSG()** number. This directive allows messages to be changed on the fly. For example, **MSG(164)** `"&Yes, &No"` can be changed to another language:

```
DEF MSG(164) = "&Oui, &Non"
```

Using Customized Messages with WindX

Use of this facility under WindX requires some additional effort by the developer; i.e., will the subprogram be running on the host or on the workstation. If the program runs on the host, it will transmit the screen drawing information to the workstation just like any other ProvideX program. If the program is to run on the workstation, the host will simply send the **MSGBOX** parameters to WindX, which in turn runs the program locally (assuming it is present).

The setup for WindX is described as follows:

- To run the host's `msgbox.gui`, set the '**MX**' system parameter. No change is required for workstation software.
- To run the workstation's `msgbox.gui`, ensure that the program exists on the workstation, then execute `[wdx]Set_param 'MX'` to set the parameter locally.

This takes advantage of the fact that ProvideX automatically sets '**MX**' based on the existence of a (user-defined) `*ext/msgbox.gui`. Simply copy the `msgbox.gui` from `*ext/system` to the `*ext` directory on the host, the system will use it and send screen drawing directives to the workstation. If it is copied (or installed) to `*ext` on the WindX workstation, the system will automatically use it, assuming it is not overridden by the host.

This customizable **MSGBOX** also takes advantage of the '**BEEP**' mnemonic.

See Also

['MX' System Parameter, p.675](#)

[DEF MSG Define Temporary Message, p.70](#)

Examples

```
1000 MSGBOX "Remove the customer record",
1000: "Confirmation Please", "? , YESNO" , X$
1010 IF X$ <> "YES" THEN RETURN
1020 REMOVE (1, KEY=K$)
```

MULTI_LINE Directive

Control Multi-Line Input


Formats

1. *Define/Create*: **MULTI_LINE** *ctl_id*, @(col,ln,wth,ht)[,ctrlopt]
2. *Remove*: **MULTI_LINE REMOVE** *ctl_id*[,ERR=*stmtref*]
3. *Disable/Enable*: **MULTI_LINE** {**DISABLE** | **ENABLE**} *ctl_id*[,ERR=*stmtref*]
4. *Lock/Unlock*: **MULTI_LINE** {**LOCK** | **UNLOCK**} *ctl_id*[,ERR=*stmtref*]
5. *Hide/Show*: **MULTI_LINE** {**HIDE** | **SHOW**} *ctl_id*[,ERR=*stmtref*]
6. *Force Focus*: **MULTI_LINE GOTO** *ctl_id*[,ERR=*stmtref*]
7. *Signal on Focus*: **MULTI_LINE SET_FOCUS** *ctl_id*,*ctl_val*[,ERR=*stmtref*]
8. *Read Current Value*: **MULTI_LINE READ** *ctl_id*,*var*\$(,mode\$)[,ERR=*stmtref*]
9. *Load Value*: **MULTI_LINE WRITE** *ctl_id*,*contents*\$(,ERR=*stmtref*)
10. *Report All Changes*: **MULTI_LINE AUTO** *ctl_id*[,ERR=*stmtref*]

Where:

@(col,ln,wth,ht) Position and size of the multi-line input region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.

contents\$ Text / contents of the multi-line input field. String expression.

ctl_id Unique logical identifier for multi-line input (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the  key) or [Negative CTL Definitions, p.817](#). Use this value with the apostrophe operator to access various [Multi-Line Properties](#).

ctrlopt Control options. Supported options for **MULTI_LINE** include:
ERR=*stmtref* Error transfer
HLP=*string*\$ Help functionality, see [AutoComplete, Calendar, p.219](#).
FMT=*mask*\$ For valid options, see [Data Format Masks, p.813](#). FMT=" ! " can be used to indicate that the field should be displayed as blank if the value is 0 zero; however, do not use FMT=" ! " if you wish to display a 0 zero being entered, or if the value being entered can have 0 as its integral component; e.g., - 0.99 or .12.
FNT="font,size[,attr]" Font name, size, optional properties. Refer to the ['FONT' Mnemonic, p.609](#) for details.
KEY=*char*\$ Hot key
LEN=*num*\$ Maximum input characters. You can use the **LEN=** option in conjunction with the **FMT=** option to limit the number of characters allowed for a **MULTI_LINE WRITE** that violates the format mask.
MSG=*text*\$ Message line
MNU=*ctl* CTL value associated with right-click menu event.
NUL=*string*\$ Empty value
OPT=*char*\$ (See [MULTI_LINE OPT= Settings, p.216](#).)



OWN=*name*\$ Name assigned for automated testing of this control.

SEP=*char*\$ Delimiter character. Hex or ASCII string value.

TIP=*text*\$ Mouse pointer message.

To change the colour, see '**TC**'= [System Parameter](#), p.689.

ctl_val CTL value to generate when focus goes to the multi-line input field.

mode\$ String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to terminate the multi-line input field, e.g., \$0D\$ for  as the **EOM** character for a 1-line multi-line input field, or \$09\$ for  to exit a multi-line input field.

stmtref Program line number or label to transfer control to.

var\$ String variable to receive the text/contents of the multi-line input field.

MULTI_LINE OPT= Settings

Available attribute/behaviour settings are listed below. Single characters may be combined. Invalid settings are ignored.

- "\$" Password entry displays dollar sign as substitute for each character entered.
- ">" Include a horizontal scrollbar.
- "!" Support for Arabic characters (right to left entry).
- "A" *Auto*. Generates a CTL value signal for every character entered.
- "B" *No border*. The multi-line input region will not have a border.
- "C" Centre the input.
- "d" Permanently disabled.
- "D" *Disabled*. Multi-line region is is grayed out and is not accessible to the user.
- "E" *Edit Mode*. Append to end of existing text (default=*Insert*)
- "F" *Full*. Generate a signal upon maximum input length.
- "G" *Global*. Keep active when focus changes to a new/non-concurrent window.
- "h" Permanently hidden.
- "H" *Hide*. Mult-line is not displayed but is accessible programmatically.
- "i" Suppress the '+I' mnemonic (implied decimal points) for a single multi-line input region.
- "I" Activate the '+I' mnemonic for implied decimal points for a single multi-line input region.
- "L" *Lock*. User can set focus, but cannot change the multi-line input region.
- "R" *Right Justify*.
- "s" *Scroll*. Allows the multi-line input region to scroll in a resizable/scrollable dialogue box.

- "t" Supports the use of the **Tab** key in the multi-line input region.
- "T" Strips trailing spaces.
- "U" Upper case: converts lower case to upper case automatically.
- "X" Signal when focus leaves the multi-line input region.
- "Z" Cursor changes to "resize" pointer if within 4 pixels from the edge of the control.

Multi-Line Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a multi-line control are described in [Chapter 7. Control Object Properties, p.706](#).

Description Use the **MULTI_LINE** directive to create and control a multi-line input region on the screen. Multi-line input is used to enter or display text. When the input is complete, the user presses a function key, **Tab**, or **Enter** for a single-line high multi-line input field. ProvideX then generates a control using the multi-line input region's associated *ctl_id*. The **AutoComplete** feature can be included to automatically prompt the user with previously-entered data. The **Calendar** feature provides a user-friendly way to enter date information into a multi-line input area.

Format 1: Define/Create Multi-Line

MULTI_LINE *ctl_id*, @(col,ln,wth,ht)[,ctrlopt]

Use this format to define or create a multi-line input field. The **FNT=** option establishes the font for multi-line input. If you omit this option, ProvideX uses the "System" default font. If you specify **FNT="*"**, standard text mode fixed font is used.

If you define a multi-line input field as occupying more than one line, ProvideX adds scrollbars and automatic word wrapping. If you define the input area as only one line high, the multi-line input region will not have a vertical scrollbar, but will scroll horizontally as required; e.g.,

```
0010 MULTI_LINE 100,@(2,14,12,6)
0020 MULTI_LINE WRITE 100,"Now is the time for all"
0030 OBTAIN *
0040 IF CTL=100 THEN MULTI_LINE READ 100,X$
```

This creates a multi-line input field that generates a **CTL=100** when its value changes and the user exits the multi-line input region.

When a popup menu is assigned to a multi-line using the **MNU=** option, a system menu consisting of Cut, Copy, Paste and Delete is automatically pre-pended to the popup menu. These menu items can be translated. The item text is derived from system message #136 from *mlfile.xx (where xx is the language code) which normally contains the value "Delete,Paste,Copy,Cut".

Besides translating `*mlfile.xx`, message #136 can also be changed using the **DEF MSG** directive; e.g,

```
DEF MSG(136) = "Supprimer , Coller , Copier , Couper , "
```

There are no spaces between the items, and the trailing comma is required. If no popup menu is assigned using **MNU=**, then the system menu is supplied by the OS, and the above does not apply.

AutoComplete

When the **AutoComplete** feature is applied to a multi-line control, it will predict text as the user is entering it. When the control is accessed, users are prompted with the previously-entered words/phrases or entries already in a data file as soon as they start to type. This is a particularly useful tool for applications that require repetitive data entry. The **AutoComplete** feature will be disabled when **MULTI_LINE** is used for a password field.

AutoComplete may be set up using the **HLP=** option as follows:

```
MULTI_LINE ctl_id, @(col,ln,wth,ht)),HLP="[AutoComplete]parameters$"
```

Enter the following parameter list (*parameters\$*) separated by semicolons:

AUTOPURGE=YES|NO

Automatically purges expired records. Expired words or phrases are only purged when the multi-line is accessed. Default is **NO**.

DATAFILE=*path\$* Name of keyed file that contains the words/phrases. This file should be resident and accessible on the local workstation.

EXPIRED=*num* Number of days a given record will be used before expiry. If this is not set or set to 0, the words/phrases do not expire.

FIELD=*num* Field that is being displayed.

KNO=*num* Key number to be used.

LENGTH=*num* Maximum number of characters that will be displayed.

OFFSET=*num* Starting position within the field to be displayed.

PREFIX=*string\$* Prefix that will be used for searching matching words/phrases.

READONLY=YES|NO

ProvideX does not automatically update the key file when user enters a new word and/or phrase. Default is **YES**.

The '**AutoComplete\$** property can also be used (with the above parameter list); e.g., `AutoComplete$="DataFile=ProvideX.dat;ReadOnly=NO"`.



Note: If the user is going to be using live data that is not supposed to be updated, ensure that **READONLY=YES**.

Existing parameters will not be reset unless they are set specifically when the **AutoComplete\$** property is run. If you need to reset all parameters in the list, run the **AutoComplete\$** property set to a *null* string.

The list of previous multi-line entries is stored in an keyed file that is accessible to the application. **AutoComplete** will be based on the internal key of the key file and the key has to be case insensitive for the property to be working correctly. If the key is case sensitive, all lower case key will be ignored.

Example:

In the following code, `MultiLineA` is defined using a **MULTI_LINE** directive declaration and `MultiLineB` is defined using the **AutoComplete\$** property. They both use the same key file.

```
0010 LET F$="AutoComplete.dat"; ERASE F$,ERR=*NEXT
0020 KEYED F$,[1:1:30:"C"],0,-40
0030 LET MultiLineA=1000; LET MultiLineB=1001
0040 MULTI_LINE MultiLineA,@(25,4,20,1),HLP="[AutoComplete]
      DATAFILE=AutoComplete.dat;READONLY=NO"
0050 MULTI_LINE MultiLineB,@(25,8,20,1)
0060 LET MultiLineB'AUTOCOMPLETE$="DATAFILE=AutoComplete.dat;READONLY=NO"
0070 ESCAPE
```

The result will appear similar to the address edit box in a web browser. Initially, the key file is empty, so nothing will happen when the user types. When the user enters new text in the multi-line, each word/phrase will be saved in the keyed file. If the user tries to type the same word/phrase again, it will find a match from the list before they finish typing.

Client-Server Behaviour. In a client-server environment, the file used by the auto-complete logic to store and/or retrieve data must be on the client machine by default.

If you wish to retrieve data from a file on the server, you must use additional program logic to accomplish this. First, set the **AutoCTL** property of the **MULTI_LINE** with a `ctl_id` to be generated when content is needed for loading values in the auto-complete dropdown. Monitor events to trap the CTL value as a signal to execute logic to build the list for the auto-complete dropdown. Assign the list to the **MULTI_LINE**'s **AutoValue\$** property. This will cause the dropdown to be loaded for selection.

Calendar

The **CALENDAR** feature provides a user-friendly way to enter date information into a multi-line input area. When it is applied to a **MULTI_LINE**, a button will be added to the control that can be clicked to invoke a *month calendar* popup. This will allow users to pick a date to be inserted automatically into the multi-line input area. The format of the date inserted is based on the formatting rules of the **DTE() Function, p.422**. The **CALENDAR** feature may be set up using the **HLP=** option as follows:

```
MULTI_LINE ctl_id, @(col,ln,wth,ht)),HLP="[Calendar]parameters$"
```

Enter the following parameter list (*parameters\$*) separated by semicolons:

```
CALENDAR=YES|NO
```

YES turns on the calendar support. **NO** turns it off. Default is **NO**.

- CONTENTS=string\$** Text or graph appearing on the button, default is `{!DATE}`.
- DTE=date\$** Date formatting rules. Default is based on the **DTE()**. *Semi-colon* cannot be part of this parameter (if used, the string following will be ignored). Date code should include `% percent`; however, if not used, input will be parsed based on format provided. If a time formatting string is included, the current time is used.
- HEIGHT=num** Height of button. Defaults to height defined for **MULTI_LINE**.
- SHOWBUTTON=YES|NO**
YES shows the calendar button. **NO** hides it. Default is **YES**.
- WIDTH=num** Width of the button. Default width is equal to the height defined for the **MULTI_LINE**; i.e., the default size is a *square*.

The '**Calendar\$**' property can also be used (with the above parameter list); e.g.,
`Calendar$="CALENDAR=YES;DTE=%Y%M%D;Contents={!Stop}Stop;Width=10"`.

When invoked, the top left corner of the calendar button will be aligned with the top right corner of the multi-line input area. If parameters are not specified, the default graph will be `{!DATE}` and the width and height of the button equal to the height of the control itself.

Use a keyboard shortcut or mouse click to invoke the calendar:

1. When the multi-line input area or the calendar button has focus, press **Shift** - **F2**. In this case `CTL=-6` is suppressed.
2. When button control has focus, press **SPACEBAR** or **ENTER**.
3. Mouse is clicked on the calendar button.

The calendar disappears automatically:

1. When a date is selected and placed in the multi-line input area.
2. When **Esc** is pressed.
3. When the user clicks anywhere outside the button.

When the button is clicked, no EOM value will be generated, as the button is considered part of the **MULTI_LINE** directive and is handled internally. When a date is inserted into the multi-line input area, no EOM value will be generated unless `OPT="A"` has been set and focus is on the **MULTI_LINE** control.

When the **MULTI_LINE** is hidden or disabled, the calendar button should also be hidden or disabled, and **Shift** - **F2** will not display the calendar control.

Example:

The following code sample invokes the **CALENDAR** feature using both methods (via the `HLP=` string and the `Calendar$` property).

```
0010 PRINT 'DIALOGUE'(0,0,100,50,"test",OPT="*Z")
0020 PRINT '4D','CS'
0030 LET A=1000
0040 MULTI_LINE A,@(10,10,30,1),HLP="[CALENDAR]CALENDAR=YES"
0050 LET B=1001
```

```

0060 MULTI_LINE B,@(10,20,30,1)
0070 LET B'CALENDAR$="CALENDAR=YES;DTE=%Y %M1 %D;Contents=Enter
      Date;Width=10"
0080 PRINT "A: ",A'CALENDAR$
0090 PRINT "B: ",B'CALENDAR$
0100 OBTAIN (0,SIZ=1)'ME',A$, 'MN',; LET C=CTL; LET E$=HTA(EOM); PRINT
      "CTL=",C," EOM=",E$
0110 IF C=4 THEN ESCAPE
0120 GOTO 0100

```

Format 2: Remove Multi-Line

MULTI_LINE REMOVE *ctl_id*[,ERR=*stmtref*]

This format to deletes a multi-line input field.

Format 3: Disable/Enable Multi-Line

MULTI_LINE {**DISABLE** | **ENABLE**} *ctl_id*[,ERR=*stmtref*]

Use the **MULTI_LINE DISABLE** format to gray out a multi-line input region so that it will be visible but *inaccessible* to users. To reactivate it, use **MULTI_LINE ENABLE**.

Format 4: Lock/Unlock

MULTI_LINE {**LOCK** | **UNLOCK**} *ctl_id*[,ERR=*stmtref*]

Use the **MULTI_LINE LOCK** or **UNLOCK** formats to prevent or allow access to the multi-line input field. Use a **LOCK** to limit the user's permissions to "view only" for data such as notes and instructions.

Format 5: Hide/Show

MULTI_LINE {**HIDE** | **SHOW**} *ctl_id*[,ERR=*stmtref*]

With the **MULTI_LINE HIDE** format, the multi-line remains active, but is not displayed. It is still accessible programmatically. Use the **SHOW** format to restore the display and user access.

Format 6: Force Focus

MULTI_LINE GOTO *ctl_id*[,ERR=*stmtref*]

Use the **MULTI_LINE GOTO** format to reactivate and force focus to a multi-line input field, ready for the next user action.

Format 7: Signal on Focus

MULTI_LINE SET_FOCUS *ctl_id*, *ctl_val*[,ERR=*stmtref*]

Use the **MULTI_LINE SET_FOCUS** format to define an alternate CTL value to generate whenever focus shifts to the multi-line input region.


Format 8: *Read Current Value*

MULTI_LINE READ *ctl_id,var\$[,mode\$][,ERR=stmtref]*

If the user enters more than one line of text, ProvideX will delimit each line, either using the system **SEP** character or the character you specify in the **SEP=** option of a directive. By default, multi-line input fields auto-wrap text, so the **SEP** character will only be returned on the **READ** if the user inserts a carriage return.

On the **READ**, the *mode\$* variable will identify the method use to end multi-line input. It will contain:

\$09\$ for  used to exit or

\$0D\$ for  used to end a 1-line multi-line input field.

Once this value is read, it is reset to \$00\$.

Use the **MULTI-LINE READ** format to read the contents of the multi-line input field.

MULTI-LINE READ can also be used to retrieve window caption; e.g.,

```
MULTI_LINE READ 0 ,A$
```

A\$ will contain window caption after command is executed.

Format 9: *Load Value*

MULTI_LINE WRITE *ctl_id,contents\$[,ERR=stmtref]*

When you write string expressions into the multi-line input, you must use a delimiter set by either the **SEP=** or the **DLM=** option.

Format 10: *Report All Changes*

MULTI_LINE AUTO *ctl_id[,ERR=stmtref]*

Use the **MULTI_LINE AUTO** format to generate a CTL value on any keystroke in the multi-line.

MULTI_MEDIA Directive Control Multimedia Interface

Format **MULTI_MEDIA** *command*[\$[,*var*[\$[,*ctl_val*]]]

Where:

command\$ String containing the multi-media command to execute. String expression.

ctl_val String variable to receive any response or error message.

var\$ Optional CTL code to be generated when a NOTIFY signal is returned by the multimedia system. Numeric expression.



Note: This directive only functions under WindX or Windows.

Description

Use the **MULTI_MEDIA** directive to pass commands to the Windows Multimedia Control Interface (MCI). These strings can contain commands that will play WAV files, MID files, AVI files or control various multi-media devices. Typical commands include:

Command	Function/Purpose
open filename	Opens the specified file and loads the required drivers. You can append an optional 'alias name' to change the name of the file and make controlling the file easier.
close filename	Closes the specified files and releases the drivers.
close all	Stops and closes all files.
play filename	Plays the specified file. The file will be opened (if not already opened), played, and closed automatically.
rewind filename	Rewinds the file.
stop filename	Stops the playback.

Examples

The following examples illustrate the different uses for the **MULTI_MEDIA** directive.

Example 1:

```
MULTI_MEDIA "open C:\Documents and Settings\Default User\Application
Data\files\demo.avi alias video"
```

Example 2:

```
0010 ! Close all previous Multi_media commands for this session
0020 MULTI_MEDIA "close all"
0030 !
0040 ! Assign an Alias to the Wave file for use in the Play command
0050 MULTI_MEDIA "open C:\WINDOWS\Media\tada.wav alias wavefile"
0060 !
0070 ! Issue the Play command requesting notification of a CTL=100
0080 ! after the wave file has finished
0090 MULTI_MEDIA "play wavefile notify",100
0100 !
0110 ! Wait for the CTL=100
0120 OBTAIN X$
0130 IF CTL=4 THEN STOP
0140 IF CTL<>100 THEN GOTO 0110
0150 PRINT "Multi_media command finished"
```

Future references to the file could simply use the alias. Two options can be appended to the play command:

NOTIFY send the CTL value when completed.
WAIT wait for the playback to complete.

If an error occurs during processing, it will be returned in the *return\$* variable.



Note: Full documentation of all available MCI commands is beyond the scope of this manual. Refer to other (Microsoft) sources for complete documentation on the MCI system.

NEXT Directive

End FOR Loop

Format **NEXT** [*var*]

Where:

var Optional control variable to be incremented or decremented. This variable must match the current **FOR** variable.



Note: Refer to [FOR..NEXT Loop While Incrementing, p.134](#), for complete syntax.

Description

When ProvideX encounters the **NEXT** directive, the control variable is incremented or decremented and, if the current value does not exceed the ending value, control is returned to the directive following the corresponding **FOR**.

If a variable is used in the **NEXT** directive, it must match that of the currently active **FOR**. If no **FOR** is active or the variables do not match, ProvideX returns an Error #28: No corresponding FOR for NEXT. The control variable can be omitted from the **NEXT** directive because the increment/decrement of the **FOR var** is assumed automatically. However, the **NEXT var** is useful for readability purposes, especially if it appears within a nested loop structure.

See Also

[FOR..NEXT Loop While Incrementing, p.134](#)

Example

```
0010 FOR I=1 TO 10
0020 PRINT I,
0030 NEXT ! NEXT I, since I is current
-:RUN
1 2 3 4 5 6 7 8 9 10
```

NEXT RECORD Directive**End SELECT Statement**Format **NEXT RECORD***Note:* Refer to [SELECT Query Records, p.299](#), for complete syntax.**Description**

Use the **NEXT RECORD** directive to end a **SELECT** directive. As each record is read, ProvideX processes any logic you include following the **SELECT** directive up to the **NEXT RECORD**. When ProvideX encounters a **NEXT RECORD** statement with no records found for a nested **SELECT**, it will locate the corresponding **SELECT** statement.

Example

```
0010 SELECT IOL=0100 FROM "CUST_FILE",KNO=1 BEGIN "ABC CO" END "NEW CO" WHERE
0010:CITY$="CLARENDON"
0020 PRINT REC(IOL=0100)
0030 NEXT RECORD
0100 IOLIST CUST$,NAME$,ADDR1$,ADDR2$,CITY$,PROV$,POSTAL$,INV_DT$,AMT,TERMS,
0100:DUE_DT$
0110 PRINT "DONE"; END
-:run
123460
ACME LTD.
SUITE 1900
2000 1ST ST.
CLARENDON
ON
KOKOKO
0
```

OBTAIN Directive*Get Hidden Terminal Input*

Format **OBTAIN** (*chan* [, *fileopt*]) *varlist*

Where:

chan Channel or logical file number of a terminal or indexed file from which to obtain input.

fileopt Supported file options (see also, [File Options, p.810](#)):

ERR=*stmtref* Error transfer

HLP=*string*\$ Help message identifier

LEN=*num* Limit on input size

SIZ=*num* Number of characters to read (number of screen columns)

TBL=*stmtref* Data translation table

TIM=*num* Maximum time-out value in integer seconds

varlist Comma-separated list of variables, literals, mnemonics, **IOL=** options, and/or location functions '@(...)'.

Description Use the **OBTAIN** directive to issue prompts to terminal devices and to process responses (the user's input). The file reference should be to a terminal, but you can use an indexed file. If you include literals or expressions in this directive, ProvideX treats them as prompts for the user.

You can include format masks, as in `A$=STR(.01:"0.00")`. If you omit the format mask for a numeric in the **OBTAIN** statement, **'DP' Decimal Point Symbol** and **'TH' Thousands Separator** system parameters are ignored for European decimal settings.



Note: **OBTAIN** performs in the same way as the **INPUT** directive except that the user's input is not echoed on the screen. This can be useful for such applications as passwords.

See Also

[ACCEPT Read Single Keystroke, p.28](#)

[INPUT Get Input from Terminal, p.160](#)

['ME' Mnemonic, p.620](#)

['BI' Mnemonic, p.590](#)

[Data Format Masks , p.813](#)

Example

```
0010 OBTAIN 'CS' ,@(5,5) ,"Enter your password:",C$
RUN
Enter your password:
```

The password, "TEST" in this example, is not echoed, but C\$ will return the value.

```
- :?C$
TEST
```

ON EVENT Directive

Event Processing

Format

1. *COM Control via OOP Object*: **ON EVENT FROM** *com_id* **PROCESS** *oop_id*
2. *COM Control via CTL Event*: **ON EVENT** *evtname\$* **FROM** *com_id* **PREINPUT** *ctl_id*
3. *Remove CTL Event*: **ON EVENT** *evtname\$* **FROM** *com_id* **REMOVE**

Where:

com_id Numeric CTL value of a Windows COM object.

ctl_id Numeric CTL signal to generate (preinput) when a given event occurs.

evtname\$ Event name, maximum 255 characters.

oop_id Numeric identifier of an OOP object.

Description

This directive is used to activate support for individual COM events for use in ProvideX applications. This is a feature of the ProvideX *Event Handling Interface* and *Component Object Model* (COM), an industry-standard technology used by applications to expose methods, properties, and events to development tools, macro languages, and other applications. For complete information on this subject, refer to the document *Automation in ProvideX*, available for download from the ProvideX website www.pvx.com.

Format 1: COM Control via OOP Object

ON EVENT FROM *com_id* **PROCESS** *oop_id*

This format is used to register a numeric OOP object identifier (*oop_id*) to service a given COM control (*com_id*). The OOP object identifier is stored in a read-only property of the COM object called '**PvxEvents**'. A comma-separated list of the events that are supported by the COM object is available by querying '**PvxEvents\$**' via the [Apostrophe Operator, p.823](#). Supported events are prefixed with a plus sign (+) while unmanaged events are prefixed with a leading minus sign (-).



Note: The list of events reported in '**PvxEvents\$**' is only available after the **ON EVENT FROM** *com_id* **PROCESS** *oop_id* has been executed. This behaviour is intended to minimize communication with the interop layer when event support is not required.

An invalid *com_id* generates an Error #65: Window element does not exist or already exists. An invalid *oop_id* generates an Error #95: Bad Object Identifier. Other errors, such as when a COM object does not support events, will generate an Error #88: Invalid/unknown property name and, if available, place a description of what caused the error in the '**PvxError\$**' property for the COM object, as well as in **MSG(-1)**.

An *oop_id* of 0 zero deactivates event processing for the current COM object. Dropping a ProvideX OOP object deactivates event processing for all COM objects associated with the OOP object.

Issuing a subsequent **ON EVENT** directive for a COM control discontinues event processing for the first ProvideX OOP object, then activates it for the new OOP object (provided the new *oop_id* contains a non-zero value).

Formats 2 and 3: *COM Control via CTL Event*

ON EVENT *evtname*\$ **FROM** *com_id* **PREINPUT** *ctl_id*

Generate Event. The **PREINPUT** option is used to generate a ProvideX CTL event whenever the identified COM event occurs. This format simplifies the event interface by eliminating the need to create an OOP object to manage events. This format will work across WindX. The event process does not have access to any event parameters as it will not be running in-line when the event occurs.

ON EVENT *evtname*\$ **FROM** *com_id* **REMOVE**

Remove Event. The **REMOVE** option stops the process of generating a CTL signal for the specified event. However, the removal of an assigned event will have no effect on currently queued events.

See Also

[DEF OBJECT Define Object, p.71](#)

[Apostrophe Operator, p.823](#)

[External Components, User's Guide.](#)

[Data Integration, User's Guide.](#)

ON..GOSUB Directive Conditional Subroutine Execution

Format `ON num GOSUB stmtref,stmtref,...`

Where:

num Value determines the location to which to transfer. Numeric expression.
Integer range: -32768 to +32767.

stmtref Program line number or label to transfer control to.

Description Use the **ON..GOSUB** directive to transfer control to a subroutine at one of the statement references listed, based on the numeric value supplied:

num <=0, control is transferred to the first statement specified.

num =1, control is passed to the second statement number specified.

num =2, control is passed to the third, and

num =3... *nnn*, control is passed sequentially.

If *num* is greater than the number of *stmtref*'s supplied, the last *stmtref* is assumed.

See Also [GOSUB.. Execute Subroutine, p.141](#),
[RETURN Subroutine/Function Return, p.291](#)

Examples `0020 ON X GOSUB 0100, 0200, 0300, 0400, 0500`

<i>ON...</i>	<i>GOSUB</i>
X <= 0	Transfers to 0100
X = 1	Transfers to 0200
X = 2	Transfers to 0300
X = 3	Transfers to 0400
X >= 4	Transfers to 0500

ON..GOTO Directive Conditional Transfer of Control

Format **ON** *num* **GOTO** *stmtref,stmtref,...*

Where:

num Value determines the location to which to transfer. Numeric expression.
Integer range: -32768 to +32767.

stmtref List of statement references. Use program line numbers or labels to which to transfer control conditionally.

Description Use the **ON ... GOTO** directive to transfer control to one of the statements listed based on the value you provide:

num <=0, control is transferred to the first statement specified.

num =1, control is passed to the second statement number specified.

num =2, control is passed to the third, and

num to *nnn*, control is passed sequentially.

If *num* is greater than the number of *stmtref*'s supplied, the last *stmtref* is assumed.

See Also [GOTO Transfer within Program, p.142.](#)

Examples 0020 ON X GOTO 0100, 0200, 0300, 0400, 0500

ON...	GOTO
X <= 0	Transfers to 0100
X = 1	Transfers to 0200
X = 2	Transfers to 0300
X = 3	Transfers to 0400
X >= 4	Transfers to 0500

OPEN Directive

Open for Processing

Formats

1. *Open File/Device Channel*: **OPEN** (*chan*[,*fileopt*])*string*\$
2. *Open for Read-Only Mode*: **OPEN INPUT**(*chan*[,*fileopt*])*string*\$
3. *Open Locked*: **OPEN LOCK** (*chan*[,*fileopt*])*string*\$
4. *Open Locked and Pre-Cleared*: **OPEN PURGE** (*chan*[,*fileopt*])*string*\$
5. *Open Static Keyed File Read Only*: **OPEN LOAD** (*chan*[,*fileopt*])*string*\$
6. *Open for Use in Object (OOP)*: **OPEN OBJECT** (*chan*[,*fileopt*])*string*\$

Where:

<i>chan</i>	Channel or logical file number to be assigned to a file or device.
<i>fileopt</i>	Supported file options (see also, File Options, p.810): BSZ=num Buffer size (in bytes) ERR=stmtref Error transfer IOL=iolref Default IOList ISZ=num Open file in binary mode (see limits below) KEY=pswd\$ Password to open file (see PASSWORD Directive, p.239) NBF=num Dedicated number of buffers OPT=char\$ <i>MS Windows</i> open options REC=name\$ Record prefix (REC=VIS(string\$) can also be used) When using the ISZ= option, the following limits apply: ISZ=1 indicates greater than 2GB access, ISZ>1 is limited to 2GB.
<i>string</i> \$	Name of the file or device to open. Refer to Special Files and Devices, p.737 , Special Command Tags, p.769 and COM Ports and Serial Devices below.
<i>stmtref</i>	Program line number or label to transfer control to.

Description

Use this directive to open a file or device and assign a logical file (channel) number to it. The *string*\$ expression can include a specialty filename or file tag; e.g., ***MEMORY***, **[RPC]**, etc. It may also contain a port ID for direct serial communication.

You can normally have a maximum of 127 files open at any time in a ProvideX session (less under some operating systems). In the extended file access mode '**XF**' you can open up to 65000 files, subject to OS limitations.



Note: The actual number of files you can open at any one time depends on operating system parameters. Consult your system configuration information for details on how to increase the number of files you can open.

Opening Devices. Use a colon (:) at the end of the *string*\$ to open a port number directly; i.e., LPT1 and LPT1: are considered to be the same device. For information on opening devices, see [COM Ports and Serial Devices](#) (below).

Special command file tags are used to modify paths and filenames for specific I/O in ProvideX. These are listed and described under [Special Command Tags, p.769](#).

For information on accessing built-in virtual files, devices, and interfaces using the **OPEN** directive, see [Special Files and Devices, p.737](#).

CLOSE, BEGIN, START, STOP, BYE, QUIT, RELEASE, END directives will close currently-open channels.

Error Messages on OPEN

If you use a channel that is already **OPEN**, ProvideX returns an Error #14. If you try to **OPEN** a file that doesn't exist, it returns an Error #12: File does not exist (or already exists). In some circumstances (e.g., in trying to **OPEN** a printer twice), an Error #0 is generated.

```
-:KEYED "TEST",[1:1:6],,128
 -:OPEN (5)"TEST"
 -:OPEN (5)"TEST" ! Not okay: channel 5 is already in use
Error #14: Invalid I/O request for file state
 -:OPEN (4)"TEST" ! Okay - "TEST" file can be OPEN on two channels
 -:OPEN (30)PRINTER$
 -:OPEN (10)PRINTER$ ! Not okay:
Error #0: Record/file busy
```

File OPEN Options

Use the **OPT=** parameter to define options for Windows COM ports. For details, see [COM Ports and Serial Devices](#) below.

Use the **IOL=** option to define a standard IOList to be used while the file is open. ProvideX will use this IOList for all subsequent file **READ, WRITE, EXTRACT, or FIND** statements where you do not explicitly supply variable lists. You can also use the **REC=** option to supply a prefix to be added to all the variables in the **IOL=** specification.

If the **ISZ=** option is used, ProvideX opens the file in binary mode. That is, ProvideX makes no attempt to analyze the file structure or contents. All subsequent access to the file / channel is done as if the file is an indexed file with a record size equal to the value set in the **ISZ=** option.

With **ISZ=1**, a **READ RECORD** directive will return 1 byte at a time (each logical record is 1 byte). For **ISZ=1024**, the data is returned 1024 bytes at a time, with the first 1024 bytes in **IND=0**, the next 1024 bytes in **IND=1** and so on. You can gain access to the file sequentially or by using the index. A negative value notifies ProvideX to return up to the given number of bytes.

If you use **ISZ=-1**, ProvideX will not append record terminators to output lines printed to a file without a hanging comma. (Normally ProvideX appends a line feed to the end of such output lines in UNIX, or a carriage return and line feed in Windows and to the end of every record written to a serial file.)

If you use the **BSZ=** option, file access will be buffered in a buffer equal to the size you set for the option.

COM Ports and Serial Devices

OPEN can be used to open a COM port for direct serial communication and to gain access to special serial devices, such as label printers, weigh scales, modems, and access card readers. Some devices, such as modems, involve two-way communications, where you can set up tasks such as background processes to monitor the port. Other devices, such as label printers, use one-way serial communication; e.g., in **PRINT** statements.

Windows serial devices:

OPEN (*chan,fileopt,OPT=string\$*)"port:"

Use this format in Windows to open a serial device for direct access. You can specify communications port settings for COM ports by adding **OPT=string\$**; e.g., in **OPEN** (1,OPT=settings\$) "COM2:". The string variable in the **OPT=** option would contain values representing baud rate, parity, data bits, stop bits and xon/xoff flow control.

The following attributes and format apply to the **OPT=** option:

OPT="baud_rate,parity,data_bits,stop_bits[,flow_rate]"

Where

- baud rate** Valid range is 300 to 115200
- data_bits** Use one of three valid values: 1, 7, or 8
- flow_rate** *Optional.* Valid values include: **x** for xon/xoff software flow or **p** for RTS/CTS physical/hardware flow. (Omit the value for "none")
- parity char.** One of three alpha characters: **N** = *None*, **O** = *Odd*, **E** for *Even*
- stop_bits** Use one of the valid values: -1 (minus one), 0, 1, 1.5, 2.

For example, the *setting\$* assigned in the following example are for the serial communications attributes: baud rate (9600), parity (*n=none*), data bits (8), stop bit (1) and flow rate (*x=xon/xoff switch*).

```
12540 let setting$="9600,n,8,1,x"
12550 let printer=hfn
12560 open (printer,isz=1,opt=setting$,err=13000)"COM2:"
12570 print (printer)"Title"
```



Note: If the **OPT=** option is omitted from the communications settings, ProvideX will default to the setup in the Windows Control Panel.

UNIX/Linux serial devices:

OPEN (*chan[,fileopt]*)*dev_path\$*

When you **OPEN** the serial device, the port is opened with the current characteristics. To change the settings, you can either set line characteristics at the OS level before starting ProvideX or use an **INVOKE** directive after you open the device:

```
invoke "stty 38400 -IXANY IXON IXOFF ... </dev/tty2A"
```

WindX Tip. While it is possible to open a serial device on the client PC by using the [WDX] tag, we recommend against doing this. Instead, we advise you to use a **CALL** from the host to a [WDX] subprogram on the remote client PC. Design your subprogram to open and control the device and its settings. You can return your results to the host via your **CALL**'s parameters. For more information see [WDX] [Direct Action to Client Machine, p.801](#).

OPEN with PREFIX FILE Definition

You can have two fields in a prefix file data record. (A prefix file is a special Keyed file that contains information to be used for dynamic translations of files when they are opened.) The first of the two fields is the path/filename of the real file to open. The second is an options field. ProvideX uses any options in this field as the **OPT=** values when opening the real path/filename. When opening a filename assigned in a prefix file directive, you can include additional **OPT=** values in the **OPEN** directive to have ProvideX append these as additional options for the true file being opened (i.e., the filename in the prefix file record).

Given a prefix file record containing:

```
Key="GLMAST" , DATA RECORD=" [ ODB ]DSN;TABLE"+sep+"KEY=field1 "
```

If you `OPEN(chan) "GLMAST"` then internally ProvideX will

```
OPEN (chan , OPT="KEY=field1" ) " [ ODB ]DSN;TABLE "
```

If you `OPEN (chan , OPT="REC=somedata") "GLMAST"` then internally ProvideX will `OPEN (chan , OPT="KEY=field1 ; REC=somedata") " [ODB]DSN;TABLE "`

See Also

[CLOSE Directive, p.56](#),
[OPT\(\) Function, p.495](#),
[PREFIX Set File Search Rules, p.249](#)
[PASSWORD Apply Password & Encryption, p.239](#)
[Special Files and Devices, p.737](#)
[Special Command Tags, p.769](#)

Format 1: *Open File/Device Channel*

```
OPEN (chan[,fileopt])string$
```

Use this format to open a given file or device so that a program can gain access to it; e.g., `OPEN (2 , ERR=1000) "CSTMER" .`

Format 2: *Open for Read-Only Mode*

```
OPEN INPUT (chan[,fileopt])string$
```

If you use **OPEN INPUT**, the program can't update the file (opened as read-only). Use this format to open a disk directory or to access files with read-only permissions. The **OPEN INPUT** directive under UNIX will not lock a text mode device.

Format 3: *Open Locked*

OPEN LOCK (*chan* [, *fileopt*])*string*\$

Using this format, the file is reserved for exclusive use prior to the **OPEN**. However, if another user already has the file **OPEN**, the **LOCK** format of the directive fails and ProvideX returns an Error #0: Record/file busy.



Note: Under UNIX, /dev/null and /dev/console files are not locked when opened. However, all other files are.

WindX Example:

```
00010 BEGIN
00020 PRINT
00030 !
00040 LET in_file$=%wdx$+"D:\DATA\BATCH\R0000188"
00050 LET o_file$="D:\JUNK\TEST\TST_OUT"
00060 EXECUTE "[WDX]ERASE "+o_file$+",ERR=*NEXT"
00070 EXECUTE "[WDX]SERIAL " "D:\JUNK\TEST\TST_OUT" ""
```



Note: WindX supports the use of **SERIAL** and **ERASE** commands via the **[WDX]** tag. It is not necessary to embed these commands in an **EXECUTE** directive. (If you are running a version of ProvideX earlier than Version 4.20 on a WindX PC, you may need to encapsulate these commands in an EXECUTE "[WDX]..." directive, as in lines 0060 and 0070 above.) See also [\[WDX\] Direct Action to Client Machine, p.801](#).

```
00130 LET no_file$="Y"
00140 LET in_file=HFN;
      OPEN (in_file)in_file$
00150 LET f$=FIN(in_file)
00160 CLOSE (in_file)
00165 LET pd=POS(DLM=in_file$,-1);
      IF pd<>0 \
          THEN LET inn_file$=in_file$(pd+1) \
          ELSE LET inn_file$=in_file$
00170 LET chars=DEC(f$(1,4))
00180 !
00190 LET blk_size=1024,done$="N",c=0
00200 IF chars<blk_size \
      THEN LET blk_size=chars
00210 !
00220 OPEN (in_file,ISZ=blk_size)in_file$
00230 LET o_file=HFN;
      OPEN LOCK (o_file,ISZ=blk_size)%wdx$+o_file$
00250 LET cur_byte=0,st_byte=0,se_byte=0
00260 LET r_c=blk_size
00500 ! !500
00510 ! Get Pos Of First ISA
```

```

00520 !
00530 READ RECORD (in_file,END=0700)in_rec$
00540 LET l=l+1;
        PRINT @(0,5),l
00550 WRITE RECORD (o_file)in_rec$
00555 LET by=by+LEN(in_rec$)
00556 LET tst=chars-by;
        IF tst<blk_size \
            THEN LET r_c=tst
00590 GOTO 0530

```

Format 4: *Open Locked and Pre-Cleared*

OPEN PURGE (*chan[,fileopt]*)*string*\$

If you use the **OPEN PURGE** format, the file will be locked and pre-cleared (purged) prior to the completion of the **OPEN** statement.

Format 5: *Open Static Keyed File Read Only*

OPEN LOAD (*chan[,fileopt]*)*string*\$

With the **OPEN LOAD** format, ProvideX assumes that the file is a static Keyed file and opens it for **READ** access only. ProvideX assumes that the file is a static Keyed file; i.e., no other task on the system will update this file. Whenever a portion of the file's key structure is read into memory, ProvideX keeps it in memory until you **CLOSE** the file. ProvideX gives you extremely fast access to static files by reducing the disk I/O and effectively caches the file in memory.

Format 6: *Open File for Use in Object (OOP)*

OPEN OBJECT (*chan[,fileopt]*)*string*\$

In *Object Oriented Programming*, the **OPEN OBJECT** directive indicates that a file being opened is for the exclusive use of an object. Only the object itself can alter the state of the file, and once the object is deleted, the file is automatically closed. Any external attempt to alter the state of the file returns Error #13: File access mode invalid.

The file is not closed on an external **BEGIN**. It will only be closed when the object is deleted, or by an explicit **CLOSE** (*chan*) from within the object. **FFN** and other file functions will not see the file while outside of the object thus cannot effect its position or other characteristics.e; however, system variables like **HFN** and **CHN** will reflect that the file is open, and some functions (**PTH**, **FIN**, **FIB**, etc.) can be used to query file attributes.

Example:

```

DEF CLASS "Customer"PROPERTY CUST_NO$, NAME$, ADDR$, CITY$, SALESMAN$, AMT_OWING
LOCAL FILE_NO
!
FUNCTION FIND(X$)

```

```
ENTER C$
    READ (FILE_NO,KEY=C$) ! Loads all the variables
    RETURN 1
!
FUNCTION NEXT()
    READ (FILE_NO,END=*NEXT); RETURN 1
    RETURN 0!
FUNCTION UPDATE()
    WRITE (FILE_NO); RETURN 1
END DEF
!
ON_CREATE:
    FILE_NO = HFN; OPEN OBJECT (FILE_NO,IOL=*)"ARCUST"
    RETURN
```

The file is opened in the ON_CREATE. There is no need to worry about closing the file since ProvideX does it automatically.

For further information, see **Data Integration**, p.275 in the *User's Guide*.

PASSWORD Directive*Apply Password & Encryption*

Formats

1. *On Program*: **PASSWORD** *pswd*\$
2. *On Common Password to All Programs*: **PASSWORD** *[,*pswd*\$]
3. *On Data File - Required*:
PASSWORD (*chan*[,**ERR**=*stmtref*]) *pswd*\$ **REQUIRED FOR OPEN**
4. *On Data File - Read Only*:
PASSWORD (*chan*[,**ERR**=*stmtref*]) *pswd*\$ **REQUIRED FOR WRITE**
5. *On Data File - Required & Encryption*:
PASSWORD (*chan*[,**ERR**=*stmtref*]) *pswd*\$ **REQUIRED FOR OPEN AND ON DATA**
6. *On Data File - Read Only & Encryption*:
PASSWORD (*chan*[,**ERR**=*stmtref*]) *pswd*\$ **REQUIRED FOR WRITE AND ON DATA**
7. *Copy Password to Data File from Data File*:
PASSWORD (*chan1*[,**ERR**=*stmtref*]) **FROM** (*chan2*[,**ERR**=*stmtref*]) [,**ERR**=*stmtref*]
8. *Remove Password from Data File*: **PASSWORD** (*chan*[,**ERR**=*stmtref*]) **REMOVE**

Where:

* Asterisk defines a password as common to all programs.

chan Channel or logical file number.

pswd\$ Password for program/data file protection. String expression limited to 240 characters.

stmtref Program line number or label to transfer control to.

Description

Use the **PASSWORD** directive to assign/remove passwords to/from programs and data files.



Important: When encryption is enabled on a data file, all key and data blocks will be encrypted; therefore, routines that attempt to parse a passworded file in binary mode will not function correctly. This includes the file recovery utility, *UFAR.

Formats 1 and 2: *Assign or Remove Passwords on Programs*

The formats described in this section assign/remove password protection on *programs*. Passworded programs cannot be listed or edited in ProvideX in any way unless the correct password is used.

PASSWORD *pswd*\$

Apply to Program. To assign a password, load the program, enter the **PASSWORD** directive followed by the new password *pswd*\$, then save the program; e.g,

```
->LOAD "MYPROG"
->PASSWORD "CAT"
->SAVE "MYPROG"
->LOAD "MYPROG"
->LIST
Error #52 -- Program password protected
->DELETE 10
```

```

Error #52 -- Program password protected
->PASSWORD "CAT"
->LIST
0010 REM...
0020 ... etc.

```

Before changing a password, you must reload the program and enter the **PASSWORD** directive followed by the previously assigned password. At this point, you can either *change* the password by entering **PASSWORD** (again) followed by a new string, or *remove* password protection by entering **PASSWORD** (again) followed by a null string.

PASSWORD *[,*pswd*\$]

Apply Password Common to all Programs. Use the *asterisk* * to denote a common password. ProvideX will apply a common password automatically to all previously passworded programs when they are loaded and to all new programs.

Formats 3, 4, 5, and 6: Assign Password to Data File

The formats described in this section assign password protection to *data files*. A **KEY=pswd**\$ option is required to **OPEN** a passworded file. In order to define/change a password, you must have exclusive access to the file and it must be empty. The encryption feature is only available for VLR and EFF files.

Use one the following syntax formats to assign a password to a data file:

PASSWORD (*chan1*[,*ERR=stmtref*]) *pswd*\$ **REQUIRED FOR OPEN**

Required for Open indicates that the correct password is always required on a open.

PASSWORD (*chan1*[,*ERR=stmtref*]) *pswd*\$ **REQUIRED FOR WRITE**

Required for Write indicates that the correct password is required for write access, but it is not required for read-only access.

PASSWORD (*chan1*[,*ERR=stmtref*]) *pswd*\$ **REQUIRED FOR OPEN AND ON DATA**

Required for Open and on Data indicates that the correct password is always required and that the data is encrypted.

PASSWORD (*chan1*[,*ERR=stmtref*]) *pswd*\$ **REQUIRED FOR WRITE AND ON DATA**

Required for Write and on Data indicates that the correct password is required for write access but it is not required for read-only access, and that the data is encrypted.

The following table outlines the usage, access level, and encryption associated with each syntax format used to assign a password to a data file:

PASSWORD Format	Access Level	Without Password			With Correct Password			Encrypted
		Open	Read	Write	Open	Read	Write	
OPEN	0	No	No	No	Yes	Yes	Yes	No
WRITE	1	Yes	Yes	No	Yes	Yes	Yes	No
OPEN AND ON DATA	2	No	No	No	Yes	Yes	Yes	Yes
WRITE AND ON DATA	3	Yes	Yes	No	Yes	Yes	Yes	Yes

An internal password queue records passwords for successfully opened files and checks when an attempt is made to open a passworded file without specifying a **KEY=** clause or when a null **KEY=** value is supplied. The password stored in the queue is used if an entry exists for that file. The number of entries to keep in the queue is controlled by the '**PQ**' [System Parameter, p.680](#). The ability to distinguish between an invalid password and a non-existent password is provided by means of the '**PE**' [System Parameter, p.679](#).

Due to the fact that all key and data blocks are encrypted, routines that attempt to parse a passworded file in binary mode will not function correctly. This includes the file recovery utility *UFAR.

Prompting for Password

ProvideX includes a generic program called `get_pswd` that will prompt for a password when **KEY=** is invalid or missing when a passworded file is opened. ProvideX checks the existence of the `get_pswd` program in the `*ext` subdirectory first, and then in `*ext/system` if the former is not found. This feature also allows the developer to customize the interface. As the prompt will be handled by a called program, it is also WindX-aware.

An embedded I/O (EIO) processing entry point called **Get_Password** provides the ability to prompt the user for a password based on logic associated with the EIO program. Provided the EIO program is valid and the entry point **Get_Password** exists, it will be used instead of the generic `*ext/system/get_pswd` or custom `*ext/get_pswd`. As the file is not in an **OPEN** state at the point when the entry point is called, the LFO and LFA values do not contain meaningful information. For this reason, the name of the file will be passed in the fourth parameter, normally referred to as **Value\$**.

Examples:

```
->KEYED "MyFile",[1:1:10],0,0
->OPEN LOCK (1)"MyFile"
->PASSWORD (1)"ABC" REQUIRED FOR OPEN
->CLOSE (1)
->OPEN (1,KEY="ABC")"MyFile"
->WRITE (1)"Record A"
->LOCK (1)
->PASSWORD (1)"XYZ" REQUIRED FOR OPEN
Error #13: File access mode invalid
->PURGE (1)
->PASSWORD (1)"XYZ" REQUIRED FOR OPEN
->CLOSE (1)
->OPEN (1,KEY="XYZ")"MyFile"
```

Monitoring Attempts

TCB(68) reports the number of attempts which have been made to prompt for the password. This value is incremented prior to ProvideX calling the embedded I/O or `get_pswd` routine so the first attempt will have a **TCB(68)** value of 1 (one).

By default, the first three attempts to access a passworded file using an invalid password will result in a prompt to re-enter the password. The fourth attempt generates an `Error #53: Invalid password`. This behaviour is controlled via the **'PP' System Parameter, p.680**.

Password Error Reporting

The following error conditions will be trapped:

`Error #13: File access mode invalid.`

Attempt to apply or remove a password when the file is in read-only mode, not locked, or not empty.

`Error #14: Invalid I/O request for file state.`

Attempt to apply a password to an un-opened channel.

`Error #17: Invalid file type or contents.`

Attempt to apply a password to a non-Keyed file or to encrypt a non-VLR formatted file.

`Error #46: Length of string invalid.`

Attempt to assign a password longer than 240 characters.

`Error #53: Invalid password.`

Attempt to open a file using invalid password.

`Error #61: Authorization failure.`

Password record failed the internal CRC check.

Format 7: Copy Password to Data File from Data File

PASSWORD (*chan1* [,ERR=*stmtref*]) **FROM** (*chan2* [,ERR=*stmtref*]) [,ERR=*stmtref*]

PASSWORD FROM allows a password from one file to be copied directly to another file *without prompting the user for the password*. Its use is primarily for rebuilding data files on the fly.



Note: An `Error #13: File access mode invalid` will occur if the destination file has an existing password and an `Error #53: Invalid password` is generated when the source file does not contain a password.

Format 8: Remove Password from Data File

PASSWORD (*chan* [,ERR=*stmtref*]) **REMOVE**

This format removes password protection from a data file. In order to remove a password, you must have exclusive access to the file, and it must be empty.

See Also

'EL' System Parameter, p.663

'PE' System Parameter, p.679

'PP' System Parameter, p.680

'PQ' System Parameter, p.680

OPEN Open for Processing, p.232

PERFORM Directive *Call Subprogram, Share Variables*

Format **PERFORM** *subprog\$*[:*entry\$*][,**ERR**=*stmtref*]

Where:

subprog\$ Name of the subprogram to execute. String expression.

;entry\$ Optional name of starting line label that is entry point in the subprogram.

stmtref Program line number or label to transfer control to.

Description The **PERFORM** directive saves the current program state, then transfers control to a subprogram. All variables are made common between the initiating program and the subprogram. When the subprogram terminates, control returns to the initiating program at the directive following the **PERFORM** directive.



Note: All variables that are changed or created during execution of the *performed* subprogram will be returned to the initiating program.

You can specify an optional entry point in the subprogram. To do this, append a semicolon and the starting label name (*;entry\$*) to the subprogram name; e.g., `PERFORM "SUBPROG; STARTING_LABEL"`. After the subprogram is loaded, ProvideX internally issues a **GOTO** directive using the label as a statement reference and starts execution there. Use this feature to create subprograms to act as "libraries" (i.e., multiple stand-alone routines, each starting at its own entry point).

The execution of the subprogram is normally terminated with an **EXIT**, however, the **END** or **STOP** directives may be used in its place.

Subroutine within a Subprogram

PERFORM can also access *subroutines* externally via entry points in the called program. In this case, the **RETURN** statement that is used to terminate the subroutine in a subprogram will automatically return control to the initiating program. This feature allows the same chunk of code to be accessed internally (**GOSUB**) as well as externally (**PERFORM**).

See Also

CALL Transfer to Subprogram, *p.40*,
RUN Transfer and Execute a Program, *p.294*
END Halt Program Execution, *p.113*
EXIT Terminate Subprogram and Return, *p.124*
STOP Halt Program Execution, *p.330*
GOSUB.. Execute Subroutine, *p.141*
RETURN Subroutine/Function Return, *p.291*
 Called Procedures, *User's Guide*.

Example*This is the calling program:*

```

0180 LET Z=1,X=2,A$="Cat",B$="Pig*****Dog",V=9
0190 PRINT "Values before PERFORM:",@(26),Z,X,A$,B$,@(45),V
0200 PERFORM "ABCDEF;TEST",ERR=9000
0210 PRINT "Values after PERFORM: ",@(26),Z,X,A$,B$,@(45),V,ZZ$
0220 STOP
->RUN

```

```

Values before PERFORM:      1 2CatPig*****Dog 9
TEST in subprogram ABCDEF
In PERFORM                  1 2CatPig*****Dog 9
Change :                    6 7Pig*****Cat***** 4 I'm new, from ABCDEF
Values after PERFORM:      6 7Pig*****Cat***** 4 I'm new, from ABCDEF

```

This is the subprogram:

```

0040 TEST: PRINT "TEST in subprogram ABCDEF"
0060 PRINT @(8), "In PERFORM",@(26),Z,X,A$,B$,@(45),V
0070 LET Z=6,X=7,A$="Pig",B$="*****Cat*****",V=4,ZZ$=" I'm new, from ABCDEF"
0080 PRINT @(8), "Change : ",@(26),Z,X,A$,B$,@(45),V,ZZ$
0090 EXIT

```



POP Directive

Premature Exit from Stack

Format POP

Description Use this directive to pop, or clear, the top entry on the stack. The **POP** directive performs the same operation as an **EXITTO** directive for terminating a **FOR/NEXT** or **GOSUB** operation except that it does not transfer control (to a statement reference), but continues with the next statement in the execution sequence..



Note: Use the '**POP**' mnemonic (not the **POP** directive) if you want to pop a child window superimposed on a parent window.

See Also [EXITTO End Loop, Transfer Control, p.125](#)
['POP' or 'WR' Mnemonic, p.633](#)

Example ProvideX will generate an Error #28: No corresponding FOR for NEXT if it encounters a **NEXT** directive after you **POP** a **FOR** loop. In this example, the **POP** directive is used to terminate a **FOR** loop, leaving the **GOSUB**'s **RETURN** entry on the stack. (No Error #28 is generated.)

```
0100 GOSUB SCAN_FOR_IT
...
1000 SCAN_FOR_IT
1010 FOR I=1 TO 1000
1020 IF X$(I)="MOOCOW" THEN POP; RETURN
1030 NEXT
1040 PRINT "NOT FOUND"; RETURN
```

POPUP_MENU Directive

Create Popup Menu

Format **POPUP_MENU** [*@(col,ln)*], *list\$*, [*strvar\$* | *numvar*]

Where:

@(col,ln) Numeric expressions. The optional column and line position on the current window where the menu will appear.

list\$ Menu structure and elements. String expressions. Maximum 2047 elements. A bitmap/icon can be included for each element.

numvar Name of variable that receives the numeric value of the selection made.

strvar\$ Name of variable that receives the string containing the selection sequence of characters selected.

Description Use the **POPUP_MENU** directive to create and process floating menus. A popup menu "pops up" on top of the current window when you right-click the mouse while the pointer is over a control, such as a button, multi-line, or list box. Once a popup menu is created and assigned to a control, it remains invisible until the user right-clicks the mouse button over the enabled component. In a popup menu, a default option may be specified.

Popup Menu Definition

POPUP_MENU definition uses a format similar to the [MENU_BAR Directive, p.202](#):

- Each menu group is enclosed by square brackets.
- Each item in a group is separated by a comma.
- Each item's selection character (hot key) is preceded by an '&' (ampersand).
- Each sub-menu group is prefixed with its item ID.

Numeric values are assigned to each entry in the menu definition. By default, the first entry has the value '1', the second '2', etc. This value can be overridden by placing an equal sign and numeric value after the menu item text; e.g.,

```
"F: [&Open, &Save, &Quit=4]"
```

Quit would have a value of four. In addition, the starting numeric value can be changed or reset by placing a **#** followed by the new starting point within the menu definition string outside of group definitions. When setting a new starting value, the next menu selection will be assigned a number that is one more than the starting point; e.g.,

```
"[&File, &Quit], #1000, F: [&Open, &Save, &Rename]"
```

This defines **File** as one, **Quit** as two, **File-Open** as 1001, **File-Save** as 1002, and **File-Rename** as 1003. Accelerator keys should be unique for each selection on a given sub-menu within a group. While the use of duplicate accelerator keys is permitted, it is difficult to determine which selection was made by using just the character strings.

Menu items can be *disabled*, displayed in *bold* or with a *checkmark*, by placing a "D", "B", or "C" after the = (equal sign) and before the value to return; e.g.,

```
m$=" [&One=1 , &Two=C2 , &Three=D3 , &Four=BC4 ] "
popup_menu m$ , answer$
```

The resulting menu shows "Two" *checked*, "Three" *disabled*, and "Four" *bolded and checked*.

Using Images

Images can be included for each item in the menu. Enclose the image name in curly braces and place it in the menu definition just prior to the specific item text; e.g.,

```
"- [&File] , F: [&Open=1001 , { !Stop } &Stop=1002 ] "
```

Use a leading exclamation point (!) to identify the image as internal, or specify the relative path and filename to access an image file that is external. **POPUP_MENU** also supports the use of *two-tone effects* as described under the **MENU_BAR Directive, p.204**. For more information on options available for displaying internal/external images and the recognized image file types, see **Images and Icons, p.153** in the *User's Guide*.

Popup Menu Assignment

A **POPUP_MENU** can be associated with buttons, check boxes, drop boxes, grids, list boxes, multi-lines, radio buttons and tristate boxes. The directives that control these objects, include a **MNU=** option that defines the CTL number that will be generated when the user right-clicks on the control.

See Also

[MENU_BAR Directive, p.202](#)
[BUTTON Control Button, p.34](#),
[CHECK_BOX Control Check Box, p.47](#)
[DROP_BOX Control Drop Box, p.96](#)
[GRID Control Grid, p.143](#)
[LIST_BOX Control List Box, p.178](#)
[MULTI_LINE Control Multi-Line Input, p.215](#)
[RADIO_BUTTON Control Radio Button, p.265](#)
[TRISTATE_BOX Control Tristate Box, p.344](#)

Example

This popup menu returns a numeric value assigned by the menu bar definition

```
0020 print 'CS', ; list 0030,
0030 check_box 100, @(50,10,10,2)="{ }&Popup Menu" , opt="P" , mnu=101
0040 obtain *
0050 if ctl=4 then stop
0060 if ctl<>101 then goto 0040
0070 mdef$=" [&File, &Edit, E&xit=4] , F: [&Open=10 , &Save=D20] ,
      E: [C&ut=30 , &Copy=40 , &Paste=50 ] "
0080 popup_menu @(58,12) , mdef$ , x
0090 print "Selected: " , x
0100 if x=4 then stop
0110 goto 0040
```

PRECISION Directive

Change Current Precision

Formats

PRECISION *num* [FOR OBJECT]

Where:

num Precision to which to round. Numeric expression. Integer range: 0 to 18, or use -1 to switch to scientific notation.

FOR OBJECT *Object Oriented Programming* (OOP) - Optional keywords for setting the default **PRECISION** for all subsequent method invocations within the current object instance, except for those that have **CLASS** definitions that specifically declare a **PRECISION** to use (preserving encapsulation).

Description

Use the **PRECISION** directive to change the number of digits ProvideX maintains to the right of the decimal point. (Internally, ProvideX makes all calculations using 18 digits of accuracy but when numeric data is output the value is rounded and returned using the number of digits set in the **PRECISION** directive.)

The precision setting is also used in conjunction with the **ROUND** directive. If rounding is enabled (default mode), values assigned to variables through the **LET** directive will be rounded to the **PRECISION** *num* specified.



Note: Precision is automatically reset to two (2) by the following directives: **BEGIN**, **CLEAR**, **END**, **LOAD**, **RESET**, **STOP**, **RUN**.

The **FLOATING POINT** directive overrides the **PRECISION** directive. **PRECISION -1** can also be used to have ProvideX switch to floating point (scientific notation). Use any another numeric value in the **PRECISION** directive (e.g., **PRECISION 2**) to cancel scientific notation. See **Data Integration**, p.275 in the *User's Guide*.

See Also

[FLOATING POINT Switch to Scientific Notation, p.133](#),
[ROUND Control Rounding, p.293](#)
[PRC\(\) Function, p.503](#)
[PRC System Variable, p.569](#)
['PD'= System Parameter, p.679](#)

Examples

```
0010 FOR A=0 TO 6
0020 PRECISION A
0030 PRINT 1/3,
0040 NEXT A
0050 PRINT " DONE"; STOP
RUN
0 0.3 0.33 0.333 0.3333 0.33333 0.333333 0.3333333 DONE

0060 PRECISION 7
0070 PRINT 1/3, @(15),"PRECISION: ",TCB(14)
0080 BEGIN ! Resets precision to 2
0090 PRINT 1/3, @(15),"PRECISION: ",PRC
0100 END
RUN
0.33333333 PRECISION: 7
0.33 PRECISION: 2
```


PREFIX Directive

Set File Search Rules

Formats

1. *Set Prefix 0:* **PREFIX** [search_string\$]
2. *Set Any Prefix Number:* **PREFIX** (num)[search_string\$]
3. *Set Program Retrieval Rules:* **PREFIX PROGRAM** [search_string\$]
4. *Specify Search Rules in Keyed File:* **PREFIX FILE** [filename\$]

Where:

filename Name of a variable-length Keyed file containing data records that define the locations of specific files (in effect, a lookup or translation table).

num Numeric value between 0 and 9 which defines the **PREFIX** entry to use. If you omit this value, the default is zero.

search_string\$ One or more pathname prefixes (search locations) you want ProvideX to search when attempting to find files / programs. Optional string expression.

Use a null string, **PREFIX [(num)]**"" , or omit the string, **PREFIX [(num)]** to have ProvideX reset your **PREFIX** to null.

When you have more than one search location in a **PREFIX** directive, the different locations must be space-separated. If you need to include a space within a directory name, then that directory location must be enclosed in double quotation marks; e.g.,

```
PREFIX "C:\Tmp\ C:\Usr\ "C:\Program Files\ ""
```

Description

Use the **PREFIX** directive to define a series of search paths to be inserted in front of all relative file references used in **OPEN / LOAD / RUN / CALL / PERFORM** directives. (ProvideX opens files with absolute paths directly, without performing a search.)

Each prefix can contain 0 *zero* or more search locations. (A search location is either a directory or a disk/directory pair.) You can specify up to 10 differently-numbered prefixes (i.e., **PREFIX** (0) through **PREFIX** (9)) as well as a **PREFIX PROGRAM** and a **PREFIX FILE**. See the descriptions of the various formats, below.

See Also

[ProvideX Search Rules, p.251](#),
[Equal Signs for Matching, p.251](#),
[Asterisks as PREFIX Wildcards, p.252](#),
[\[LIB\] Tag, p.781](#)

Using the PREFIX Directive in the ProvideX User's Guide.

Format 1: Set Prefix 0 (Zero)

PREFIX [*search_string*]

If no numbered prefix is specified, then the **PREFIX** command affects PREFIX (0) by default. For example, PREFIX "C:\TMP" would apply to PREFIX (0).

Format 2: Set Any Prefix Number

PREFIX (*num*)[*search_string*]

Use this format to define a maximum of 10 numbered prefix table entries (range 0 to 9) to set file search rules. Each table entry can contain multiple paths; e.g.,

```
PREFIX ( 4 ) "PGMS\CST\CASHRCPT\ \PGMS\MGMT\MISC\ \PGMS\SALES\ "
```

Format 3: Set Program Retrieval Rules

PREFIX PROGRAM [*search_string*]

Use the **PREFIX PROGRAM** format to define the search location(s) ProvideX will search first when it attempts to **LOAD/RUN/CALL/PERFORM/SAVE** programs; e.g.,
PREFIX PROGRAM "\other\pgm\tst\ "

If you omit the prefix string or use a null string (PREFIX PROGRAM " "), ProvideX will reset the program prefix to null. Access to program libraries can also be defined using the **[LIB]** tag; e.g., PREFIX PROGRAM "[LIB:/usr/myappl/proglib]".

Format 4: Using Paths in Keyed File

PREFIX FILE [*filename*]

Use the **PREFIX FILE** format to specify the special Keyed file that contains information to be used for dynamic translations of files when they are opened in your applications. Any file you define as a **PREFIX FILE** must be a variable-length Keyed file or ProvideX returns Error #17: Invalid file type or contents.

If you define a **PREFIX FILE**, then ProvideX searches this special Keyed file for a search location whenever it encounters an **OPEN** command with a relative filename, using *filename* as the key to the **PREFIX FILE**. (Remember that ProvideX opens filenames with absolute paths directly, without performing a search.) Normal **PREFIX** search rules still apply after a filename has been located in the **PREFIX FILE**.

When you write to this special Keyed file, treat the filename from your program's **OPEN** command as the Key. The data record for each key contains the true filename you want opened instead. (ProvideX retrieves the data record internally with a **READ RECORD**.)

```
KEYED "someloc.dat",25
OPEN(chan)"someloc.dat"
WRITE RECORD(chan,key="myfile")"c:\tmp\myfile"
CLOSE(chan)
.....
PREFIX FILE "someloc.dat"
OPEN(chan)"myfile ! Internally this becomes OPEN(chan)"c:\tmp\myfile" instead
```

ProvideX Search Rules

The ProvideX default is to search all prefixes, in the following order:

OPEN Directive	LOAD/RUN/CALL/PERFORM/SAVE Directives
1. PREFIX FILE , if set; replaces pathname then continues sequence.	1. PREFIX FILE , if set; replaces pathname then continues sequence.
2. Current Directory; if ' CD ' system parameter is set.	2. Program Cache.
3. PREFIX 0 to 9.	3. Current Directory; if ' CD ' system parameter is set.
4. PREFIX PROGRAM , if set.	4. PREFIX PROGRAM if set.
5. Current Directory; if ' CD ' system parameter <i>not</i> set.	5. PREFIX 0 to 9
	6. Current Directory; if ' CD ' system parameter <i>not</i> set.

The **PREFIX** search rules apply not only to files being found, but also to files being created. ProvideX creates files in the first location that is permitted by the **PREFIX** rules. If '**CD**' (Search Current Directory) is on, then all files are created in the current directory (the first permitted location). If the '**CD**' system parameter is off, then ProvideX creates the file in the first location permitted by the search rules above.

Windows search rules are used to find DLLs (i.e., not **PREFIX** search rules or current directory).

Use the **ENABLE** and **DISABLE** directives to control which of the numbered prefixes ProvideX will use in the search. (While scanning prefixes 0 to 9, ProvideX ignores any prefix that is disabled.)

Note that the initial check for **PROGRAM** cache checks for a match against the original filenames. Thus, if you used **CALL "ABCD"** and you had previously loaded a program with the same name, ProvideX would use the one in cache. This eliminates the directory searches involved, but if you have duplicate program names in your system, it is possible to get the wrong one, for instance, if you **CALL "ABCD"**, change the directory / prefix, then **re-CALL "ABCD"**. If this happens for duplicate program names in your system, either clear the cache or do not use it.

For more information, refer to [DISABLE Disable Use of Prefix Table Entry, p.92](#), [ENABLE Re-Enable Use of Prefix Table Entry, p.110](#), and the '**CD**' System Parameter, [p.658](#).

Equal Signs for Matching

Equal signs in a **PREFIX** search string have special meaning. Each character of the filename that corresponds by position to an equals sign will be used to form a subdirectory name to be used in the search. For instance, if you include 1 equals

sign, ProvideX will interpret that to mean that the first character of *filename*\$ is also the subdirectory name. If you include 2 equals signs, it will take the first 2 characters as matching the subdirectory name, and so on.

ProvideX automatically finds the location to retrieve or create files by looking first for a subdirectory with a name matching, character-by-character, in sequence, the portion of the filename that corresponds to the equals signs. This allows you to sort files into subdirectories based on automated substitution of the first few characters of your filename, and accelerates the search for and/or creation and saving of files in subdirectories.

Example:

```
PREFIX (4) "C:\MYAPP\==\" . . .
```

If a directory entry in a **PREFIX** contains equals signs as shown in the example above, ProvideX evaluates the initial two characters as matching characters and uses them as the name of the subdirectory where the search will commence for relative file references. In the following example:

```
PREFIX "C:\MYAPP\==\"
OPEN (1) "ARHIST"
```

ProvideX evaluates the filename ARHIST as C:\MYAPP\AR\ARHIST for purposes of the initial search.

Asterisks as **PREFIX** Wildcards

You can also use * and ** as wildcard characters to support the use of filename extensions without modifying your code. This feature was added to the ProvideX language as of Version 4.20 primarily for Windows 2000 users, since the Microsoft Certification rules for Windows 2000 require that all files have file extensions. With the wildcard characters, you can rename files on disk with a common file extension without modifying the program code.

Using a Single Asterisk ()*

If the **PREFIX** directive includes a single star plus a specified extension as a filename, ProvideX inserts the filename from your **OPEN** command in place of the *asterisk* and searches for the filename with the added prefix; e.g.,

```
PREFIX "c:\somedir\*.PRG"
OPEN (chan) "FOOFOO"
```

In the example above, ProvideX scans the disk for "c:\somedir\FOOFOO.PRG" and opens that file if found. If FOOFOO.PRG is not found, ProvideX attempts to find and open a file named "FOOFOO".

If the filename in the **OPEN** command already includes an extension, no substitution will occur; e.g.,

```
PREFIX "c:\somedir\*.PRG"  
OPEN(nahc) "MyFile.Dat"
```

In this case, ProvideX does not add the .PRG extension when it executes the search to find and open MyFile.Dat.

*Using Double Asterisks (**)*

If the **PREFIX** directive includes two stars plus a specified extension, ProvideX inserts the filename from your **OPEN** command in place of the *asterisks* and searches first for the filename with the extension specified in the prefix; e.g.,

```
PREFIX "c:\somedir\**.PRG"  
OPEN(chan) "FOOFOO"
```

In the example above, ProvideX scans the disk for "c:\somedir\FOOFOO.PRG" and opens that file if found. If FOOFOO.PRG is not found, ProvideX attempts to find and open a file named "FOOFOO".

If the filename in the **OPEN** command already includes an extension, ProvideX adds the additional extension specified by the **PREFIX** directive when it searches for the filename; e.g.,

```
PREFIX "c:\somedir\**.PRG"  
OPEN(chan) "FOOFOO.PRG"
```

In this case, ProvideX scans for "c:\somedir\FOOFOO.PRG.PRG". However, if FOOFOO.PRG.PRG is not found, ProvideX attempts to find and open a file named "FOOFOO.PRG".

PREINPUT Directive

Place Data in Input Queue

- Formats
1. *Preinput Value & CTL*: **PREINPUT** *string\$[,ctl_val]*
 2. *Preinput CTL Only*: **PREINPUT** *ctl_val*
 3. *Jump the Queue*: **PREINPUT NEXT**

Where:

ctl_val Value to be placed in CTL when the input is processed. Numeric expression.

string\$ String expression. Placed in the input queue for the user's terminal.

Description Use the **PREINPUT** directive to have your program prime the input buffer for the user's terminal with a value. You can issue more than one **PREINPUT** directive, with the messages queued. You can also use this directive to have subprograms respond to **INPUT** statements in the **CALLing** or subsequent programs.

See Also [INPUT Get Input from Terminal, p.160](#),
[CTL System Variable, p.557](#).

Format 1: *Preinput Value, CTL*

PREINPUT *string\$[,ctl_val]*

This pre-inputs the *string\$* expression and sets the optional CTL value to be returned. In this format, the first **PREINPUT** data will be processed first (FIFO).

Format 2: *Preinput CTL Only*

PREINPUT *ctl_val*

You can add the **PREINPUT** CTL value alone to the queue.

Format 3: *Jump the Queue*

PREINPUT NEXT

When you use this format, the **PREINPUT** entry goes to the front of the queue (LIFO).

PRINT Directive

Display Information

Formats `PRINT [(chan,fileopt)]varlist .. or .. ? [(chan,fileopt)]varlist`

Where:

- `?` ProvideX accepts the question mark `?` as a substitute for **PRINT**.
- `chan` Channel or logical file number of the target device (terminal or printer) or serial file for a display or print job. The user's terminal is always defined as 0 *zero*. If omitted, the channel number defaults to 0.
- `fileopt` Supported file options (see also, [File Options, p.810](#)):
ERR=stmref Error transfer
TBL=stmref Record number.
- `stmref` Program line number or label to transfer control to.
- `varlist` Comma-separated list of variables, literals, expressions, [Mnemonics](#), **IOL=** options, and/or location functions '@(...)'. Include [Data Format Masks](#) to define how data is to be displayed.

Description Use the **PRINT** directive to format and send printable data to a terminal, printer or file. This instruction processes mnemonics and positioning information. If the print statement ends with a trailing comma, the output from a subsequent **PRINT** directive will continue on the same line. Otherwise, it will start at the first column of the next line. If you omit a *format mask* for a numeric in the **PRINT** statement, **'DP' Decimal Point Symbol** and **'TH' Thousands Separator** system parameters are ignored for European decimal settings. ProvideX accepts the question mark `?` as a substitute for **PRINT**.



Note: Because of the potential conflict between the function **AND()** and the logical operator **AND**, there is a problem when the syntax processor tries to parse the statement `PRINT AND(41, 42)`. As a work around, assign the result of a logical **AND** to a temporary variable or change the statement to `PRINT " "+AND(41, 42)`.

See Also [Mnemonics, p.577](#)
[Data Format Masks , p.813](#)

Examples

```
PRINT 'CS' , "Date:" , DAY , " Time:" , TIM      is the same as
? 'CS' , "Date:" , DAY , " Time:" , TIM.
```

Both result in the same date and time display on a clear screen (Date:02/22/00 Time: 8.398397). You can assign a page or screen position for **PRINT** data:

```
1010 PRINT @(5,5) , "CUSTOMER LISTING" , ! prints to screen at col 5, line 5
```

You can also print data overlaying a graphic in ProvideX, but only if all text output is sent using **'FONT'** and **'TEXT'** mnemonics (rather than as standard **PRINT** statements). In the example below, "Hello" will overlay the embedded bitmap.

```
0010 PRINT 'CS'
0020 PRINT 'PICTURE'(220,210,600,500,"!Binoculars",2),
0030 PRINT 'FONT'("MS Serif",-20); PRINT 'GREEN','TEXT'(220,210,"Hello")
```

PROCESS Directive

Call a NOMADS Panel

Formats

1. *Invoke a NOMADS Panel:* **PROCESS** "panel", "[lib]", arg_1\$, arg_2\$, ... arg_20\$
2. *Invoke a NOMADS Query:* **PROCESS** "panel", "[lib]", val\$
3. *Invoking File Maintenance:* **PROCESS** "panel", "[lib]", arg_1\$

Where:

arg_1\$... arg_20\$ List of valid arguments for a Panel Object. Optional string expressions. You can use up to 20 arguments. These arguments are accessible in the invoked panel as values in the reserved NOMADS variables **ARG_1\$** through **ARG_20\$**.

For Query or File Maintenance objects, you're limited to one argument, the value of which is accessible in the reserved variable **ARG_1\$**. The others are reserved.

lib Optional. Name of the NOMADS library containing the *panel* name. String expression. If you use null (""), NOMADS uses your currently active library.

panel Name of the NOMADS *Panel*, *Query* or *File Maintenance* object. String expression.

val\$ Starting / return value for a Query. See the examples, below.

Description

Use the **PROCESS** directive to call a NOMADS panel from a program. NOMADS returns to the program when the panel is exited. You can pass optional arguments to and from the NOMADS panel.



Note: When you use the **PROCESS** directive, ProvideX converts the statement internally into a **CALL** to *winproc (the NOMADS engine) to process the panel.

See Also

PROCESS, to *Invoke a Query Object* in the *NOMADS Reference*

Examples

The following examples illustrate the different uses for the **PROCESS** directive.

Panel Example:

In the example below, the ProvideX **PROCESS** statement invokes the SALES panel through the program and passes two arguments, SALES_ID\$ and STR(SALES_AMT), to the SALES panel to get the values:

```
PROCESS "SALES" , "LIBRARY.EN" , SALES_ID$ , STR(SALES_AMT)
```

In the NOMADS Panel Header **Pre-Display** logic, you would Execute the following assignments:

```
SALES_ID$=ARG_1$ ; SALES_AMT=NUM( ARG_2$ )
```


Query Example:

For queries that are not attached to a NOMADS control object,

```
0110 PROCESS "MY_QUERY" , " " , X$
```

When MY_QUERY runs, the value of X\$ is used to set the starting position in the file.

When MY_QUERY is exited, NOMADS passes the return value to the calling program in X\$.

PROCESS SERVER Directive *Establish Remote Server*

- Formats
1. *Identify Remote Server*: **PROCESS SERVER "server" ON "address"**
 2. *Terminate Remote Server*: **PROCESS SERVER "server" CLOSE**

Where:

address TCP address and services identifier for the server; e.g., "[tcp]192.1.1.100;15000". Alternatives to the TCP address are explained below.

server Name to be associated with a program server. Length: from 1 to 12 characters; e.g., "INVENTORY".

Description The **PROCESS SERVER** directive is used to identify a remote program server to a **CALL**ing application. If desired, "**address**" can be set to "LOCAL" which results in any RPC call to this server simply becoming a local **CALL** directive. Alternatively, the **address** can specify a pipe on UNIX, or a DLL on Windows. To *terminate* the remote server connection, use the **CLOSE** keyword.

Format 1: *Identify Remote Server*

PROCESS SERVER "server" ON "address"

Establishes the remote server with a logical name (between 1 and 12 characters in length). The server address may take one of several forms, depending on the circumstances; e.g.,

```
PROCESS SERVER "Inventory" ON "[tcp]192.1.1.100;15000"
```

TCP address and port/service identifier for the server.

```
PROCESS SERVER "Inventory" ON "LOCAL"
```

LOCAL keyword results in any **CALL** to this server becoming local.

```
PROCESS SERVER "Inventory" ON "|/u/sys001/inventory"
```

Pathname of | pipe server instead of TCP address (**UNIX/Linux only**).

```
PROCESS SERVER "Inventory" ON "[DLL]C:\Program
Files\Application\server.dll;Entry"
```

Pathname DLL and entry point instead of TCP address (**Windows only**).

Format 2: *Terminate Remote Server*

PROCESS SERVER "server" CLOSE

Terminates the remote server connection identified by the name "**server**"; e.g.,

```
PROCESS SERVER "Inventory" CLOSE
```

See Also [\[RPC\] Remote Process Control, p.797](#),
Remote Process Capability Technical Overview.

PROGRAM Directive *Create or Assign Program File*

- Format
1. *Create Program File*: **PROGRAM** *filename\$* [, *prog_size*] [, **ERR=***stmtref*]
 2. *Assign Default Program (OOP)*: **PROGRAM** "*interface_prog*"

Where:

<i>interface_prog</i>	Name of default program that contains object logic.
<i>filename\$</i>	Name of the program file to create. String expression.
<i>prog_size</i>	Ignored. Size of a program which can be contained in the file. Numeric expression.
<i>stmtref</i>	Program line number or label to transfer control to.

Description The **PROGRAM** directive can be used to create a program file or it can be used in *Object Oriented Programming* (OOP) to define a default program name intended to service an object.

Format 1: *Create Program File*

PROGRAM *filename\$* [, *prog_size*] [, **ERR=***stmtref*]

This format of the **PROGRAM** directive to create a ProvideX program file. The file size is for documentation purposes only and is ignored by the system; e.g.,

```
0010 PROGRAM "CSTUPD" , 1024 , ERR=1200
```

Program files have a special header format indicating that the file contains ProvideX object code. Program files should only be used with the **SAVE** or **LOAD** commands. Any attempt to use **READ** or **WRITE** directives with this type of file can yield unpredictable results when the program file is subsequently loaded.

If a given filename already exists, ProvideX returns an Error #12: File does not exist (or already exists).

WindX supports the use of this format via the [WDX] tag; e.g., **PROGRAM** "[WDX]somefile.ext" ... For more information, see [\[WDX\] Direct Action to Client Machine, p.801](#).

Format 2: *Assign Default Program in Object Oriented Programming*

PROGRAM "*interface_prog*"

In *Object Oriented Programming*, the **PROGRAM** directive is used to define the default program name that is going to service an object. This can be used to override the program that contains the **DEF CLASS**.

If this clause is specified for an object class:

- Whenever an object is created, the system will attempt to call the specified program at the label ON_CREATE.
- Whenever an object of this class is deleted, the system will attempt to call the specified program at the label ON_DELETE.

No error is reported if the label does not exist.

In addition, any references to program logic in a property read/write or a method definition can contain a leading semi-colon. For example, the following class definitions are effectively the same:

```
PROGRAM "Cust"  
FUNCTION Find(X$) ";LookupByName"
```

See Also

DEF CLASS Define Object Class, *p.65*
FUNCTION Declare Object Method, *p.137*
LIKE Inherit Properties, *p.174*
LOCAL Designation of Local Data, *p.197*
PRECISION Change Current Precision, *p.248*
PROPERTY Declare Object Properties, *p.261*
Data Integration, *User's Guide*

PROPERTY Directive

Declare Object Properties

Format

1. *Declare Property*: **PROPERTY** *prop1* [**OBJECT**], *prop2* [**OBJECT**], ...
2. *Specify Read Procedure*: **PROPERTY** *prop1* [**OBJECT**] **GET** *label* | **ERR**, *prop2* ...
3. *Specify Write Procedure*: **PROPERTY** *prop1* [**OBJECT**] **SET** *label* | **ERR**, *prop2* ...

Where:

- ERR** Keyword blocking reads or writes to a property following a **GET/SET** declaration.
- GET** Keyword indicating that logic is to be called when properties are *read*.
- label* Line label indicating **GET** or **SET** logic entry point.
- OBJECT** Optional keyword identifies that the property contains an object identifier to another object.
- prop1*, *prop2* .. Property names – treated like any other variable in the system. (In *Object Oriented Programming*, data elements are called *properties*.)
- SET** Keyword indicating that logic is to be called when properties are *written*.

Description

The **PROPERTY** directive is used in *Object Oriented Programming* to declare the properties that can be accessed by the application program. These properties can be treated like any other variable in the system and are accessible using the **Apostrophe Operator**, *p.823*.

A property name can be followed by a **GET label** to define the location of the logic to call whenever the property is read in the application. With **GET** in place, the specified logic issues a **RETURN** value that returns the actual value of the property to the application; e.g.,

```
PROPERTY ExtendedAmount GET Extension
```

```
...
```

```
Extension:
```

```
RETURN Quantity * Price
```

If the logic resides outside of the defining program, then the name of the program and entry point can be provided in quotes instead.



Note: If the logic required to do a read is simply a formula, then it can be inserted directly into the **PROPERTY** definition clause using an *equal sign* instead of **GET**; e.g.,

```
PROPERTY ExtendedAmount = Quantity*Price
```

Specify the **SET label** to intercept all of the property updates. With **SET** in place, the system calls the logic whenever the property is being updated and passes it the value being set; e.g.,

```
PROPERTY Quantity SET ChgQty
```

Like the **GET** option, an external program/entry point can be provided; e.g.,

```
PROPERTY Quantity SET "Invline;ChgQty"
```

Where `Invline` contains:

```
2000 ChgQty:
2010 ENTER NewQty
2020 IF NewQty = Quantity then RETURN
2030 ! .. Update inventory.. then RETURN
```

Use the keyword **ERR** to prevent the user from being able to **GET** or **SET** a property following the **GET** or **SET** declaration; e.g.,

```
PROPERTY ExtendedAmount SET ERR
```

If the property contains an object identifier to another object, specify the keyword **OBJECT** after the property *name*. When the object is deleted, ProvideX will use the **REF()** function against the object identifier to remove it (as long as it has no other references); e.g.,

```
DEF CLASS "Customer"
PROPERTY File OBJECT
```

When you delete an object whose class is `Customer`, then the system reduces the reference count of the object whose identifier is in `File` and, if it is no longer being referenced, deletes it as well.

For defining properties for an object class that are not exposed to external applications, refer to the [LOCAL Directive, p.197](#).

See Also

[DEF CLASS Define Object Class, p.65](#)
[FUNCTION Declare Object Method, p.137](#)
[Data Integration, User's Guide.](#)

PURGE Directive*Clear Data from a File*

Format **PURGE** (*chan*[,*ERR=stmtref*])

Where:

chan\$ Channel or logical file number of the file to be purged.

stmtref Program line number or label to transfer control to.

Description Use the **PURGE** directive to erase all data from the file specified. Before you can execute the **PURGE** directive, the file must be opened and locked. A file erased using a **PURGE** directive still exists in the system but contains no data. (ProvideX returns disk space to the system while preserving the lock.) Any **READ** directives for a purged file return an End-of-File status.

Using **PURGE** (or **REFILE**) is faster than deleting the file and recreating it.



Note: Under WindX, you can use EXECUTE "[WDX] . . ." to encapsulate a directive that is not supported across a WindX connection. See [\[WDX\] Direct Action to Client Machine, p.801](#).

See Also [REFILE Clear Data from File, p.278](#),
[LOCK Reserve File for Exclusive Use, p.200](#)

Example

```
0010 OPEN (2) "PRNTFL"
0020 LOCK (2)
0030 PURGE (2)
0040 WRITE (2)TIM, DAY
...
```

QUIT Directive

Terminate ProvideX Session

Format QUIT

Description Use the **QUIT** directive to terminate a ProvideX session and return to the operating system. If the **ERR** variable's value is not zero (i.e., an error has occurred) then the operating system is informed that an error has occurred within ProvideX. This allows you to do external testing of error conditions.

When you use **QUIT** in a compound statement, it must be the final directive.

See Also [BYE Terminate ProvideX, p.39](#)
[RELEASE Terminate ProvideX, p.279](#)

Examples 8050 IF CTL=4 THEN QUIT
 -:SAVE
 ->QUIT



RADIO_BUTTON Directive

Control Radio Button

Formats

1. *Define/Create*: **RADIO_BUTTON** [*]ctl_id:sub_id,@(col,ln,wth,ht)=contents\$[,ctrlopt]
2. *Remove*: **RADIO_BUTTON REMOVE** [*]ctl_id:sub_id[,ERR=stmtref]
3. *Disable/Enable*: **RADIO_BUTTON** {DISABLE | ENABLE} [*]ctl_id:sub_id[,ERR=stmtref]
4. *Hide/Show*: **RADIO_BUTTON** {HIDE | SHOW} [*]ctl_id:sub_id[,ERR=stmtref]
5. *Force Focus*: **RADIO_BUTTON GOTO** [*]ctl_id:sub_id[,ERR=stmtref]
6. *Logical Push/Release*: **RADIO_BUTTON** {ON | OFF} [*]ctl_id:sub_id[,ERR=stmtref]
7. *Read Activation Mode*: **RADIO_BUTTON READ** [*]ctl_id,var,mode\$[,ERR=stmtref]

Where

- * Optional. Use a leading asterisk to denote a *global* radio button.
- @(col,ln,wth,ht) Position and size of individual radio button. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines are for total area (box plus text/description). Use line value -1 to display radio button on the tool bar.
- contents\$ Text/pictures appearing on the radio button. Both {bitmap} and {icon} images are supported. String expression. See **RADIO BUTTON contents\$, p.267**.
- ctl_id Unique logical identifier for the radio button (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the **F4** key) or **Negative CTL Definitions, p.817**. Use this value with the apostrophe operator to access various **Radio Button Properties**.
- ctrlopt Control options. Supported options for **RADIO_BUTTON** include:
 - ERR=stmtref** Error transfer
 - FNT="font,size[,attr]"** Font name, size, optional properties
Refer to the **'FONT' Mnemonic, p.609** for details.
 - MSG=text\$** Message line
 - MNU=ctl** CTL value associated with right-click menu event.
 - OPT=char\$** (See **RADIO_BUTTON OPT= Settings; p.266**)
 - OWN=name\$** Name assigned for automated testing of this control.
 - TIP=text\$** Mouse pointer message.
To change the colour, refer to the **'TC'= System Parameter, p.689**.
- mode\$ String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to activate the button (\$01\$ for **MOUSE-CLICK** or \$0D\$ for **Enter**).
- sub_id Unique individual index. Integer, range from 1 to 255. Used in applications to identify which radio button the user selects.

- stmtref* Program line number or label to transfer control to.
- var* Numeric variable. Receives *sub_id* of currently activated radio button.

RADIO_BUTTON OPT= *Settings*:

Available attribute/behaviour settings are listed below. Some characters may be combined. Invalid settings are ignored.

- "<" *Bitmap Left*. Places bitmap left of text.
- ">" *Bitmap Right*. Places bitmap right of text.
- "^" *Drop-down*. Adds drop-down functionality.
- "*" *Default*. Defines button as default.
- "B" *Bitmap Button*. Has a bitmap whose width is divided into four images. Use this attribute to custom design buttons of any colour, style or shape by controlling the bitmap image that appears. Each of the four divisions represents what a button will look like in a particular state:
 - 1st quarter: Bitmap image when button is disabled.
 - 2nd quarter: Bitmap image when button is in normal (released) state.
 - 3rd quarter: Bitmap image when the mouse is over the button.
 - 4th quarter: Bitmap image when the button is pressed.
- "D" *Disabled*. Button is grayed out and is not accessible to the user.
- "F" *Flat*. Button shows no raised outline unless the mouse is over the button or the button is pushed.
- "f" *Flat-No Shift*. Same as "F", but will not shift when pressed.
- "G" *Global*. Keep active when focus changes to new/non-concurrent window. When using secondary commands (**REMOVE** or **SET_FOCUS**) on controls created with **OPT="G"** identify the control by prefixing the CTL value with an asterisk.; e.g.,
`RADIO_BUTTON 100:1,@(10,10,10,1)="Global",OPT="G"`
`RADIO_BUTTON REMOVE *100:1`
- "H" *Hide*. Button is not displayed but is accessible programmatically.
- "S" *Signal Only*. ProvideX generates a CTL value, but does not shift focus to the button automatically (the default), but only when focus is explicitly passed to it. Use this to have a button act like a function key.
- "s" *Scroll*. Button can scroll within a resizable/scrollable dialogue box.
- "T" *Transparent*. Button is "see-through" to window data below button area.
- "U" *Underscore*. Text is underscored.
- "V" *Hovertext*. Indicates that text will change colour when mouse is over the button.

Combined options can be used to create several different button types. The "f", "T", and "U" options provide the ability to turn buttons into *hotspots*. This allows for clickable areas on bitmaps or hyperlinked text in dialogues; e.g.,

- "VTf" Creates a general hotspot.
- "VUTf" Creates an HTML-like hotspot (e.g., URL hyperlink).
- "F^" Creates a word-style toolbar with drop list

Description

Use the **RADIO_BUTTON** directive to create and control a group of radio button control objects on the screen. A radio button group is a series of related circular/radio-knob buttons (akin to check boxes) of which only one button can be active at a time. When a user selects one of the radio buttons, that selection is activated (on) and all other related radio buttons are automatically reset to off.

RADIO_BUTTON *contents*\$

The *contents*\$ string expression defines the text/picture (bitmap or icon) to appear on the radio button. In the text, you can use an ampersand "&" preceding a character to identify it as a hot key the user can press in conjunction with the **Alt** key to activate the radio button from the keyboard; e.g.,

```
0010 RADIO_BUTTON 100:1,@(2,14,12,2)="&Daily"
0020 RADIO_BUTTON 100:2,@(2,16,12,2)="&Weekly"
0030 RADIO_BUTTON 100:3,@(2,18,12,2)="&Monthly"
```

This would create a group of three radio buttons that will each generate a CTL=100 when pressed. Their individual *sub_ids* are 1, 2 and 3. Their respective hot keys are D, W and M.

Using Images

When adding an image to a radio button, enclose the image name in curly braces. Use a leading exclamation point (!) to identify the image as internal, or specify the relative path and filename to access an image file that is external. There are no icons in the ProvideX executable and ProvideX does not support retrieving icons from either resource libraries or other system DLLs /executables. For more information on the options available for displaying internal/external images and the recognized image file types, types, see [Images and Icons, p. 153](#) in the *User's Guide*.

When you use text as well as images, the relative positions of the image and the text set their relative placement. The following are example *contents*\$ expressions:

```
"{!Add}Add" ! Displays the {!Add} bitmap in front of the text "Add"
"Delete{!Del}" ! Displays the {!Del} bitmap after the text "Delete"
```

If a string expression includes two images separated by a vertical bar inside a single set of curly brackets, the first will be displayed when the radio button is off (normal state), the second while the radio button is on; e.g., "{!Stop|C:\MYBMP\Go}.

Radio Button Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a radio button are described in [Chapter 7. Control Object Properties, p.707](#).

Format 1: Define/Create

RADIO_BUTTON [*]*ctl_id:sub_id,@(col,ln,wth,ht)=contents\$[,fileopt]*

Use this format to create a radio button. Unlike most other controls, related radio buttons share the same *ctl_id*. That is, to group common radio buttons together, you define each member of the group using the same *ctl_id* and a different *sub_id* (index). When the user makes a selection, other radio button *sub_ids* in the same *ctl_id* group are turned off or lose focus.

The group *ctl_id* value generates a CTL value whenever any radio button in the group is pressed and must be unique. Use an *asterisk* [*] as a prefix for *ctl_id* to identify the group of radio buttons as *global* (not tied to a specific window).

Format 2: Delete

RADIO_BUTTON [*]*ctl_id:sub_id[,ERR=stmtref]*

Use the **RADIO_BUTTON REMOVE** format to delete a radio button. By default all local radio buttons are deleted when a window is removed/dropped or the application issues a **BEGIN**. *Global* radio buttons can be removed manually or cleared with a **START** directive.

Format 3: Disable/Enable

RADIO_BUTTON {**DISABLE** | **ENABLE**} [*]*ctl_id:sub_id[,ERR=stmtref]*

Use the **RADIO_BUTTON DISABLE** format to gray out a radio button and make it inaccessible to the user. To reactivate it, use **RADIO_BUTTON ENABLE**.

Format 4: Hide/Show

RADIO_BUTTON {**HIDE** | **SHOW**} [*]*ctl_id:sub_id[,ERR=stmtref]*

With the **RADIO_BUTTON HIDE** format, the radio button remains active, but is not displayed. It is still accessible programmatically. Use the **SHOW** format to restore the display and user access.

Format 5: Force Focus

RADIO_BUTTON GOTO [*]*ctl_id:sub_id[,ERR=stmtref]*

Use the **RADIO_BUTTON GOTO** format to reactivate and force focus to a radio button, ready for the next user action.

Format 6: *Logical Push, Release*

RADIO_BUTTON {ON | OFF} [*]ctl_id:sub_id[,ERR=stmtref]

Use the **RADIO_BUTTON ON** format as a logical Push to make it appear that the radio button has been pressed. Use **RADIO_BUTTON OFF** to make it appear that the radio button has been released.

Format 7: *Read Activation Mode*

RADIO_BUTTON READ [*]ctl_id,var,mode\$[,ERR=stmtref]

The **RADIO_BUTTON READ** format returns the *sub_id* for the currently active radio button in your numeric variable. (The value is zero if none are active.) ProvideX returns a single-character hex value for the mode / keystroke the user chose to activate the last radio button. Possible values returned in *mode\$* can include:

\$01\$ for **MOUSE-CLICK**.

\$0D\$ for **Enter**.

\$20\$ for **SPACEBAR** (and keyboard HotKey, as in the example below).

\$00\$ when the user exits the button control.

To determine which **RADIO_BUTTON** *sub_id* has been pressed and the mode used:

```
0100 RADIO_BUTTON READ 1000,which,how$
```

See Also

[BUTTON Control Button, p.34,](#)

[CHECK_BOX Control Check Box, p.47,](#)

[TRISTATE_BOX Control Tristate Box, p.344,](#)

[Chapter 7. Control Object Properties, p.701.](#)

RANDOMIZE Directive

Set Random Key

Format **RANDOMIZE** *seed*

Where:

seed Numeric value or expression to be used as the seed by the random number generator.

Description Use the **RANDOMIZE** directive to assign a seed to be used by the random number generator. ProvideX uses the given value to define the starting point for the random number generator (system variable **RND**). Use the **RANDOMIZE** directive to have ProvideX return a repeatable random sequence.

See Also [RND System Variable, p.571](#)
[RND\(\) Function, p.513](#).

Example In the following example, ProvideX will **RANDOMIZE** 1, or create a repeatable random sequence using 1 as the seed. Then, for instance, each time the number 2 is encountered in the loop below, it will generate 0.59859266.

```

1000 FOR I=1 TO 2
1010 RANDOMIZE 1
1020 FOR J=1 TO 5
1030 PRINT RND, " ",
1040 NEXT J
1050 PRINT "END OF LOOP ",STR(I)
1060 NEXT I
-:run
0.11337858  0.59859266  0.81950925  0.76559375  0.5199119  END OF LOOP 1
0.11337858  0.59859266  0.81950925  0.76559375  0.5199119  END OF LOOP 2

```

In the example below, ProvideX will use a repeatable random sequence for indices; i.e., to create a repeatable number to use each time a particular index number is encountered.

```

0010 INPUT "Enter random seed:",N
0020 RANDOMIZE N
0030 OPEN (31)"TESTFILE"
0040 FOR I=1 TO 1000
0050 READ (31,IND=INT(RND*100),ERR=0090)R$
0060 PRINT RND, " ",R$, " ",
0070 NEXT I
0080 PRINT "DONE"; CLOSE (31); STOP
0090 ...

```

READ Directive

Read Data from File

Format **READ** (*chan*[,*fileopt*])*varlist*

Where:

chan Channel or logical file number of the file from which to read the data.

fileopt Supported file options (see also, [File Options, p.810](#)):
BSY=stmtref Traps Error #0: Record/file busy
DIR=num Direction indicator (not supported with **[WDX]** tag).
DOM=stmtref Missing record transfer
END=stmtref END-OF-FILE transfer
ERR=stmtref Error transfer
IND=num Record index
KEY=string\$ Record key. (See [Automatic Padding with KEY=Option](#))
KNO=num | name\$ File access key number (*num*) or name (*name\$*)
REC=name\$ Record prefix (**REC=VIS(string\$)** can also be used)
RNO=num Record number
RTY=num Number of retries (one second intervals)
SIZ=num Number of characters to read
TBL=stmtref Data translation table
TIM=num Maximum time-out value in integer seconds.

varlist Comma-separated list of variables, literals, and **IOL=** options.

stmtref Program line number or label to transfer control to.

Description Use the **READ** directive to read data from the file you identify by channel.

When it's read, the data will be split into one or more fields, either separated by the currently defined separator character or defined by embedded formats, with the contents of the first field placed in variable 1, the second field in variable 2, etc.

ProvideX will convert numeric data automatically on a **READ** statement when moving it to numeric variables. Numeric data converted during a **READ** directive does not use the **'DP' Decimal Point Symbol** or **'TH' Thousands Separator** system parameters for European decimal settings.

If you want to skip a field in the record, use an *asterisk* '*' as a place holder instead of a variable name. You can refer to an IOList instead of using a list of variables in *varlist*. To do this, use **IOL=iolref**. The *iolref* can be a line number or label for the line containing an IOList, or it can be a string containing a compiled IOList.

If you do not include variables in the **READ** directive, ProvideX will use the **IOL=** option (if you included one) in the **OPEN** statement for the given file. If *varlist* contains more variables than there are current fields in the record, then ProvideX initializes the additional variables to either zero (for numeric variables) or a null string (for string variables).

The **READ** directive will advance the file position to the next record (or, if you use a **KEY=** or **IND=** option, to the record you identify then) and if you use the **KNO=** option, the current access key will be changed accordingly.

ProvideX supports use of the **READ** directive with ***MEMORY***.

Automatic Padding with KEY=Option

When you use **KEY=string\$:string\$[:string\$][...]** ProvideX automatically pads key segments. This is valid only if you have Keyed files with segmented key definitions. ProvideX right-pads the key segment using \$00\$ (nulls) to the segment's full length.

The last segment in a compound key will not be padded.

Example:

```
KEYED "TEST", [1:1:5]+[2:1:6]+[3:1:8]
READ (1,KEY=A$:B$:C$)
```

is the same as

```
READ (1,KEY=PAD(A$,5,$00$)+PAD(B$,6,$00$)+C$)
```

See Also

FIND Locate and Read Data, *p.131*,
EXTRACT Read and Lock Data, *p.126*,
OPEN Open for Processing, *p.232*,
MEMORY Create & Use Memory File, *p.741*

Examples

```
0410 READ (1,ERR=1000,DOM=1200)A,B,*,*,E$
```


READ DATA Directive

Read Data from Program

Formats

1. *Read Data*: **READ DATA** *varlist*,...[,**ERR=***stmtref*]
2. *Read from String*: **READ DATA FROM** *data\$* [, *fileopt*] **TO** *varlist*

Where:

- data\$* Data to be parsed into the *varlist*. String expression.
- fileopt* Supported file options (see also, [File Options, p.810](#)):
END=*stmtref* End-Of-File transfer
ERR=*stmtref* Error transfer
REC=*name\$* Record prefix (**REC=VIS(*string\$*)** can also be used).
SEP=*char\$* Default separator character to parse data. Hex or ASCII string value. Dynamic **SEP=*** separators are not supported. If the **READ DATA** statement also contains a **REC=** clause, the **REC=** clause must precede the **SEP=** clause.
- stmtref* Program line number or label to transfer control to.
- varlist* List of variables to receive the values from the **DATA** statements. Comma-separated numeric and string variables are allowed.

Description Use the **READ DATA** directive to read data embedded in a program.

See Also [DATA Define Data Elements, p.63](#),
[RESTORE Reset Program Data Position, p.289](#),
[BEGIN Reset Files and Variables, p.32](#),
[LOAD Read Program into Memory, p.194](#).

Format 1: Read Data

READ DATA *varlist*,...[,**ERR=***stmtref*]

Use this format to transfer the values/expressions from data statements in a program to the variables identified in *varlist*. When **READ DATA** is executed, ProvideX evaluates each expression in the data statements, in order, and places the values into the corresponding variables defined in your **READ DATA** directive.

ProvideX increments an internal pointer to the next data expression during the reading of *data\$*. When the end of a data statement is reached, the next data statement in the program is used. Use a **RESTORE** directive to reset the pointer to the start of the first data statement.

Example 1:

Include the **END=** or **ERR=** option, to avoid an `Error #2` when the pointer is at End-of-File.

```
0010 DATA "Dog","Cat","Pig"
0020 DATA "Pig","Cat","Dog"
0030 READ DATA X$, END=1000
```

```

0040 PRINT X$, " | ",
0050 GOTO 0030
1000 RESTORE
1010 PRINT "DONE"; STOP
-:BEGIN
-:RUN
Dog | Cat | Pig | Pig | Cat | Dog | DONE

```

Format 2: *Read Data from String*

READ DATA FROM *data\$* [,REC=*name\$*][SEP=*char\$*] **TO** *varlist* [,ERR=*stmtref*]

You can read data from a string to initialize a list of variables or to parse a record into different IOLists.

Example 2. Initialize Variables:

You can **READ DATA** to an IOList to initialize all variables.

```

0010 ! Clear all variables in an IOLIST
0100 READ DATA FROM "" TO IOL=8000
8000 IOLIST ID$,NAME$,AMT

0010 READ DATA FROM "",REC=WORK$ TO IOL=IOL(WORK)

```

Example 3. Parse Record:

You can use **READ DATA** to parse a record into different IOLists.

```

0100 READ RECORD (1)R$
0200 IF R$(1,1)="T" THEN READ DATA FROM R$ TO IOL=8000
0200: ELSE READ DATA FROM R$ TO IOL=8010
8000 IOLIST TYPE$,TRN_DATE$,TRN_AMOUNT
8010 IOLIST TYPE$,PAY_DATE$,PAY_AMOUNT

```

READ RECORD Directive

Read Record from File

Format **READ RECORD** (*chan*[,*fileopt*])*var*

Where:

chan Channel or logical file number of the file from which to read the data.

fileopt Supported file options (see also, [File Options, p.810](#)):
DIR=*num* Direction indicator (not supported with **[WDX]** tag).
DOM=*stmtref* Missing record transfer
END=*stmtref* END-OF-FILE transfer
ERR=*stmtref* Error transfer
IND=*num* Record index
KEY=*string*\$ Record key
KNO=*num* | *name*\$ File access key number (*num*) or name (*name*\$)
REC=*name*\$ Record prefix (**REC=VIS(*string*\$)** can also be used)
RNO=*num* Record number
RTY=*num* Number of retries (one second intervals)
SIZ=*num* Number of characters to read
TBL=*stmtref* Data translation table
TIM=*num* Maximum time-out value in integer seconds.

var String variable. Receives the contents of the record.

stmtref Program line number or label to transfer control to.

Description **READ RECORD** reads a record from a file (*chan*) and returns the complete record's data portion to the string variable (*var*). A **READ RECORD** statement can be used when dealing with native-mode operating system files, when exchanging data between ProvideX and other applications, or to read a complete record, including data field separators.

The **READ RECORD** directive advances the file position to the next record (or the record you identify if you use the **KEY=** or **IND=** options) and, if you use the **KNO=** option, the current key access number will be changed accordingly.

ProvideX supports use of the **READ RECORD** directive with ***MEMORY*** (a memory-resident file or queue of records).

Example

```
0010 OPEN (1) "OLDFIL"
0020 OPEN (2) "NEWFIL"
0030 LOCK (2)
0040 READ RECORD (1,END=1000) R$
0050 WRITE RECORD (2) R$
0060 GOTO 0040
1000 CLOSE (1); CLOSE (2)
1010 END
```

Automatic Padding with KEY=Option

When you use **KEY=string\$:string\$[:string\$][...]** ProvideX will automatically pad key segments. This is valid only if you have Keyed files with segmented key definitions. Then, ProvideX right-pads the key segment using \$00\$ (nulls) to the segment's full length but does not pad the last segment; e.g.,

```
KEYED "TEST", [1:1:5]+[2:1:6]+[3:1:8]  
READ (1,KEY=A$:B$:C$)
```

is the same as

```
READ (1,KEY=PAD(A$,5,$00$)+PAD(B$,6,$00$)+C$)
```

Note that the last segment in a compound key will not be padded.

See Also

[RCD\(\) Function, p.508](#),
[*MEMORY* Create & Use Memory File, p.741](#)

REDIM Directive

Re-Dimension Array

Format **REDIM** *array_name*[\$](*subscript_1*[,*subscript_2*[,*subscript_3*]])

Where:

array_name[\$] Numeric or string variable to be dimensioned as an array.

subscript_1 1st dimensions (*minimum:maximum*) of array. Numeric expression, integers.

subscript_2 2nd dimensions (*minimum:maximum*) of array. Numeric expression, integers.

subscript_3 3rd dimensions (*minimum:maximum*) of array. Numeric expression, integers.

Description The **REDIM** directive is used to re-dimension an array without destroying data already contained within the array. All existing data is preserved in the exact same location within the re-dimensioned array. This directive will work on string or numeric arrays.



Note: Data in array elements that do not exist in the re-dimensioned array are discarded.

Example

```
DIM X[1:10]
X[5] = 5, X[6] = 6
REDIM X[-5:20,4 ]
```

X[5,0] has 5 and X[6,0] has 6 because the default second index is 0; i.e., the original dimension was effectively the same as X[1:10,0:0,0:0].

See Also

[DIM Define Arrays and Strings, p.86,](#)
[DIM\(\) Function, p.415](#)

REFILE Directive*Clear Data from File*

Format **REFILE** *filename*\$[,**ERR**=*stmtref*]

Where:

filename\$ Name of the file to clear. String expression.

stmtref Program line number or label to transfer control to.

Description Use the **REFILE** directive to erase all data in a specified file. If you use the **REFILE** directive to erase a file, the file still exists in the system but contains no data. (ProvideX returns disk space to the system while preserving the lock.) Any **READ** directives for the file will return an End-of-File message.

Using either the **REFILE** or **PURGE** directive is faster than deleting the file and recreating it.

If the given filename does not already exist, ProvideX returns an Error #12: File does not exist (or already exists). If the file is in use by another user, ProvideX returns an Error #0: Record/ file busy. Unlike the **PURGE** directive, the **REFILE** directive works if the file is not opened.

WindX supports the use of this directive via the [WDX] tag; e.g., **REFILE** "[WDX]somefile.ext" ... For more information, see [\[WDX\] Direct Action to Client Machine, p.801](#).

See Also [LOCK Reserve File for Exclusive Use, p.200](#),
[PURGE Clear Data from a File, p.263](#).
[Creating, Deleting, and Renaming Data Files, User's Guide](#)

Example 0010 REFILE "PRNTFL"! Pre-clear datafile
 0020 OPEN (2)"PRNTFL"
 0030 LOCK (2)
 0040 PRINT (2)'FF',"Date:",DAY,@(40),TTL\$
 0050 ...

RELEASE Directive*Terminate ProvideX Session*

Format **RELEASE** [*num*]

Where:

num Status to be returned to the operating system when ProvideX terminates. Optional. Numeric expression.



Note: The numeric value *num* is ignored on Windows platforms.

Description

Use the **RELEASE** directive to terminate a ProvideX session and return to the operating system. The numeric value is returned to the OS as a termination value, allowing for external testing of error conditions; e.g., by the system shell processor or using a 'C' program.

When you use the **RELEASE** directive in a compound statement, it must be the final directive.

See Also

BYE Terminate ProvideX, *p.39*,
QUIT Terminate ProvideX, *p.264*.

Example

```
12000 RELEASE 101
```



REM Directive

Remark

Formats **REM** *comments or .. ! comments*

Where:

! ProvideX accepts the exclamation point ! as a substitute for **REM**.
comments Remarks. Programmer's comments or documentation.

Description Use the **REM** directive to insert program remarks and comments (e.g., documentation and notes) in a program. The **REM** statement has no effect on the execution of the program. ProvideX treats all characters in a statement following the **REM** directive as comments. The **REM** directive is the last one processed in any statement.

Since ProvideX embeds the comment text following the **REM** directive in the compiled form of the program, remarks occupy memory space when the program is loaded and run.

Example

```
0010 REM Invoice Generation Program - INVGEN
0020 ! Written by: Kathryn Doe, Anytown, Canada
```


REMOVE Directive

Delete Record from File

Format **REMOVE** (*chan*[,*fileopt*])

Where:

chan Channel or logical file number of the file from which to read the data.

fileopt Supported file options (see also, [File Options, p.810](#)):
BSY=stmtref Traps Error #0: Record/file busy
DOM=stmtref Missing record transfer
ERR=stmtref Error transfer
IND=num Record index
KEY=num Record key

Description Use the **REMOVE** directive to delete a record from a file. This directive is not supported for all file types (i.e., not for indexed files). The **REMOVE** directive and the specific options are dependent on the specific file type.

Use of the **KEY=** option for Keyed, Direct, or Sort files is strongly recommended to identify the record being removed. Otherwise, if the key is not defined, **REMOVE** will target the last record requested. The key specified must be the primary key for the file. With multi-keyed files, ProvideX removes all related alternate keys from the file automatically when executing this directive.

ProvideX supports use of the **REMOVE** directive with ***MEMORY*** (a memory-resident file or queue of records, Version 4.10).

See Also [*MEMORY* Create & Use Memory File, p.741](#)

Example

```
0010 OPEN (20)"CSTFLE"
0020 INPUT "What customer do we delete? ",C$
0030 IF C$="" THEN STOP
0040 REMOVE (20,KEY=C$,DOM=0100)
0050 PRINT "Well that's gone.."
0060 GOTO 0020
0100 PRINT "Could not find ",C$
0110 GOTO 0020
```

RENAME Directive

Change a File's Name

Format **RENAME** *old_name* \${, | **TO** } *new_name* \$[,**ERR=***stmtref*]

Where:

new_name \$ New name for the file. String expression.

old_name \$ Name of an existing file to be renamed. String expression. To rename a program in a program library, use [\[LIB\]](#), [p.781](#).

stmtref Program line number or label to transfer control to.

TO Keyword, not case-sensitive (a comma may substitute for **TO**).

Description Use the **RENAME** directive to change the name of an existing file. The first string must contain the current name of the file. (The file must already exist.) Use the second string expression for the new name of the file.

When the 'OR' system parameter is set, the **RENAME** directive will assume that the new filename contains a fully expanded OS pathname to the same directory as the old filename.

If the path is omitted from the existing file name, the standard ProvideX search rules will apply. If the path is omitted from the new name, the renamed file will be located in the local working directory.



Restrictions: The new filename must not duplicate an existing filename. You must have exclusive access to the file you are renaming. ProvideX is subject to OS rules for the renaming of files.

If an operating system stores files by *inode* number, it may permit the renaming of active files because it is the inode that is in use and not the name associated with it. Other operating systems require that files not be in use before they can be renamed. Some operating systems do not allow you to rename a file to a different directory.



Note: This directive does not apply to any file segments for a multi-segmented file.

See Also **Creating, Deleting, and Renaming Data Files**, *User's Guide*.

Examples

```
0010 ERASE "ordlst"
0020 RENAME "ordcur", "ordlst"
0030 KEYED "ordcur", 6

RENAME "OLDFILE" TO "NEWFILE"
```

RENAME CLASS Directive Change Name of Class

Format `RENAME CLASS old_name$ TO new_name`

Where:

`new_name$` New name for the class.

`old_name$` Previously defined class to be renamed.

TO Mandatory keyword, not case-sensitive.

Description The **RENAME CLASS** directive is used in *Object Oriented Programming* (OOP) to alter the name of a previously defined class. This functionality allows the application designer to alter an existing object class easily, without having to change programs.

Example If you had an existing application with a Product class object but needed to add a new field to the object, such as Lot_Number:

```
RENAME CLASS "Product" TO "Orig_Product"  
DEF CLASS "Product"  
LIKE "Orig_Product"  
PROPERTY Lot_Number  
END DEF
```

All subsequent uses of a Product class object would be identical now to the standard Product class and also have the property LOT_NUMBER.

See Also

[DEF CLASS Define Object Class, p.65](#)

[DROP CLASS Delete Class Definition, p.102](#)

[DROP OBJECT Delete Object, p.104](#)

[LOAD CLASS Pre-Load Class Definition, p.195](#)

[STATIC Add Local Properties at Runtime, p.329](#)

[NEW\(\) Function, p.489](#)

[REF\(\) Function, p.512](#)

[Data Integration, User's Guide](#)

RENAME CONTROL Directive Change CTL Values

- Format**
1. *Single Control*: **RENAME CONTROL** *old_CTL* **TO** *new_CTL*{**HIDE** | **SHOW**} [,**ERR**=*stmtref*]
 2. *Multiple Controls*: **RENAME CONTROL** *from_to_list* {**HIDE** | **SHOW**} [,**ERR**=*stmtref*]
 3. *Restore Controls*: **RENAME CONTROL** *old_CTL* **TO** *new_CTL* **RESTORE** [,**ERR**=*stmtref*]

Where:

- HIDE** | **SHOW** Optional keyword to either hide or restore the control.
- from_to_list* String variable containing a series of CTL values with each entry consisting of 2 *binary* bytes for a *from* value and 2 *binary* bytes for a *to* value.
- new_CTL* New CTL value for the control. Numeric value.
- old_CTL* Existing CTL value for the control. Numeric value.
- stmtref* Program line number or label to transfer control to.
- TO** Keyword, not case-sensitive (a comma may substitute for **TO**).
- RESTORE** Keyword for restoring the visual state of a given control to where it was immediately prior to the last **HIDE** or **SHOW**.

Description The **RENAME CONTROL** directive allows graphical control objects to be renamed or hidden/shown in a single command. Use the second format when changing more than one control – this format is also intended for faster processing of GUI functionality when used in a client-server (WindX/JavX) application.

Examples The following can be used to hide graphical controls:

```
b1=10,b2=20,m1=30,m2=40
BUTTON b1,@(0,0,10,3)="abc"
MULTI_LINE m1,@(20,0,10,1)

! Change ctl values and hide controls individually
RENAME CONTROL b1 to b2 HIDE
RENAME CONTROL m1 TO m2 HIDE

! Change ctl values and show a group of controls
ctlvals$=bin(b2,2)+bin(b1,2)+bin(m2,2)+bin(m1,2)
RENAME CONTROL ctlvals$ SHOW
```

RENAME..INDEX Directive *Rename Keys in Keyed File*

Format `RENAME filename$ INDEX {keynumber|keyname$} TO {newkeyname$} [,ERR=stmtref]`

Where:

filename\$ Name of the file from where the key will be renamed. String expression.

keyname\$ Name of the key to rename (if assigned). String expression.

keynumber Key number (KNO value) to rename.

newkeyname\$ New key name.

stmtref Program line number or label to transfer control to.

TO Mandatory keyword, not case-sensitive.

Description The **RENAME..INDEX** directive allows you to rename, or name, keys. When you need to modify existing names or add key names to an older Keyed file, use the **RENAME..INDEX** directive to change the names.

Examples `RENAME "cstfile" INDEX 1 TO "Customer_no"`
`RENAME "cstfile" INDEX "#2" TO "Customer_Name"`

See also [ADD INDEX Add Key to Keyed File, p.29](#)
[DROP INDEX Drop Key from Keyed File, p.103](#)

RENUMBER Directive *Change Program Line Numbers*

Format **RENUMBER** [*new_line1* [, *step_val* [, *range_start* [, *range_end*]]]

Where:

- new_line1* New starting line number (default is 0010).
- range_start* Starting line number or label at which to begin renumbering. Optional. Use to define the start of an existing range in the program for renumbering.
- range_end* Ending line number or label with which to stop renumbering. Optional. Use to define the end of an existing range in the program for renumbering.
- step_val* Increment or step value between line numbers (default is 10). Numeric expression.

Description Use the **RENUMBER** directive to assign new line numbers to the current program. During the renumbering process, ProvideX adjusts all references to the lines being renumbered (e.g., **GOTO**, **ERR=** option, etc.) to reflect the new line number. By default, the complete program is renumbered. You can choose to resequence only selected ranges of the program. All references (**GOTO**, etc.) are adjusted in either case.

Use embedded **REM** statements in the program to control renumbering. If a remark line starts with a number (e.g., 4000 **REM** 4000), ProvideX uses the number as the absolute line number (assuming it is less than or equal to the current resequence number). The **LNO()** function can be used to avoid potential renumbering issues.

Example

```

->LIST
0001 FOR I = 1 TO 40
0002 READ (1,ERR=18)A,B
0003 PRINT A,B
0007 NEXT I
0010 STOP
0014 REM 100 - Error procedure
0018 EXITTO 0008
->RENUMBER
->LIST
0010 FOR I=1 TO 40
0020 READ (1,ERR=0110)A,B
0030 PRINT A,B
0040 NEXT I
0050 STOP
0100 REM 100 - Error procedure
0110 EXITTO 0045

```

In line 0110 (renumbered version) ProvideX generated line number 0045 in the **EXITTO** statement since no line number 0008 existed in the original version.

REPEAT..UNTIL Directive

Repetitive Execution

Format **REPEAT** ..**UNTIL** *expression*

expression Condition ends **REPEAT** looping when true. Numeric expression.

UNTIL Directive to end loop.

Description Use the **REPEAT** directive to create conditional looping in a program.

ProvideX executes all statements between the **REPEAT** directive and the next **UNTIL** directive. If the *expression* in the **UNTIL** directive is *false* (zero), ProvideX loops back to the directive following the **REPEAT** directive and resumes execution. If the *expression* is *true* (not zero), the loop is terminated and execution continues from the statement following the **UNTIL** directive.

The *expression* would normally include a logical operator (such as an *equals =*, *less-than* symbol *<*, or the **LIKE Operator**, [p.822](#)), but you can use any numeric expression.

Use the **EXITTO** directive to halt a **REPEAT/UNTIL** loop prematurely.

See Also

BREAK Immediate Exit of Loop, [p.33](#),
CONTINUE Initiates Next Iteration of Loop, [p.57](#)
EXITTO End Loop, Transfer Control, [p.125](#)
LIKE Operator, [p.822](#)
Loop Structures, *User's Guide*.

Example

```
0090 PRINT 'CS', "Standard TAX calculation..."
0100 INPUT "How much was your INCOME? $", CASH
0110 REPEAT
0120 INPUT "How much TAX did you pay? $", TAXES
0130 LET CASH=CASH-TAXES
0140 PRINT "You have $", CASH, " left"
0150 UNTIL CASH<=0
0160 PRINT "Okay you have paid enough."
-:RUN
Standard TAX calculation...
How much was your INCOME? $1.98
How much TAX did you pay? $1.65
You have $ 0.33 left
How much TAX did you pay? $.33
You have $ 0 left
Okay you have paid enough.
-:
```

RESET Directive

Reset Program State

Format **RESET**

Description The **RESET** directive does the following:

1. Resets **PRECISION** to the default value of 2. Resets **FLOATING POINT** to standard decimal notation. (In earlier versions of ProvideX, **FLOATING POINT** notation was left unchanged, i.e., *on* if *on* ...)
2. Sets **ERR**, **RET**, and **CTL** variables to zero.
3. Clears **FOR..NEXT**, **GOSUB** stacks.
4. Resets **DATA** position to the start of the program.
5. Re-enables rounding mode.
6. If 'RR' is set, ProvideX also performs a reset for **RUN** directives.

See Also **BEGIN Reset Files and Variables, p.32,**
CLEAR Reset Variables, p.54,
START Restart ProvideX, p.328,
'RR' System Parameter, p.684.

RESTORE Directive

Reset Program Data Position

Format **RESTORE** [*stmtref*]

Where:

stmtref Program line number or label the data read pointer is set to.

Description Use the **RESTORE** directive to change the position of the data read pointer. When a program is first loaded (or following a **BEGIN** directive), the data read pointer is set to the start of the program. As the data is read, the pointer is advanced through the various data statements in the program.

To set the data read pointer to a given location, include a statement reference in the **RESTORE** directive. Omit *stmtref* to set the data read pointer to the default position at the start of the program. ProvideX returns an End of File message when all data statements in the program have been read.

See Also [BEGIN Reset Files and Variables, p.32](#),
[DATA Define Data Elements, p.63](#),
[READ DATA Read Data from Program, p.273](#),
[LOAD Read Program into Memory, p.194](#).

Example

```
0010 DATA 1,2,3,"Cat"
0020 DATA 4,5,6,"Dog"
0030 READ DATA X,Y,Z,A$,ERR=0050
0040 PRINT X,Y,Z,A$; GOTO 0030
0050 INPUT "Do you want to see the last one again? (Y/N)",X$
0060 IF X$="Y" OR X$="y" THEN RESTORE 0020; GOTO 0030
0070 PRINT " DONE"; STOP
-:run
1 2 3Cat
4 5 6Dog
Do you want to see the last one again? (Y/N)y
4 5 6Dog
Do you want to see the last one again? (Y/N)n
DONE
```

RETRY Directive Re-Execute Failing Instruction

Format **RETRY**

Description The **RETRY** directive returns control to the statement on which the last error transfer occurred. This allows you to take corrective action and retry the statement. Use ***RETRY** to emulate this directive in a statement reference. See [Labels/Logical Statement References, p.816](#).

If an error occurs on a statement where you have used any of the **ERR=**, **DOM=** or **END=** options, or where you have used a **SETERR** directive, ProvideX does the following:

- Saves the statement number and directive where the error occurred (as a retry address).
- Passes control to the file option or **SETERR** statement reference to handle the error.
- Returns control to the **RETRY** directive using the retry address.

You can find the currently saved retry address or line reference in **TCB(11)**. ProvideX returns Error #27: Unexpected or incorrect WEND, RETURN, or NEXT (i.e., no return address on an error) if it encounters a **RETRY** directive with no previous statement saved.

The saved statement number and directive are cleared by any of the following directives: **BEGIN**, **CLEAR**, **LOAD**, **RESET**, **START**. You can also clear the retry address by executing a **RUN** directive with a program name specified.

When you use it in a compound statement, the **RETRY** directive must be the final directive. If **'RR'** is set, ProvideX also performs a reset for **RUN** directives.

See Also [RESET Reset Program State, p.288](#)
[TCB\(\) Function, p.534](#)
['RR' System Parameter, p.684](#).

Example

```

0010 OPEN (1)"CUSTFL"
0020 INPUT (0,ERR=1000)"Enter customer number: ",C
0030 READ (1,ERR=1010,KEY=STR(C:"000000"))C$,N$
0040 PRINT C$," ",N$
0050 STOP
1000 PRINT 'RB','LF',"Invalid customer. "; RETRY ! Ring bell, retry 0020
1010 PRINT 'RB',"Cannot find customer: "; GOTO 0020
-:run
Enter customer number: 123987
Cannot find customer:
Enter customer number: ABC
Invalid customer.
Enter customer number: 123456
123456 ABC MFG

```

RETURN Directive Subroutine/Function Return

Format

1. *Terminate Subroutine*: **RETURN**
2. *Terminate Procedure, Specify Value*: **RETURN** *expression*[\$]

Where:

expression[\$] Value to be returned from a multi-line function, embedded I/O procedure, or OOP method logic. String or numeric expression

Description

The **RETURN** directive is used to terminate a subroutine. This directive can also be used to pass a value back from a multi-line function, an embedded I/O procedure, or OOP method logic.

Format 1: *Terminate Subroutine*

The **RETURN** directive is used as the terminator for a **GOSUB** or **SETESC** subroutine. Control is returned to the initiating statement. If the subroutine has been accessed externally via the **PERFORM** directive, the **RETURN** statement will both terminate the subroutine, and return control to the initiating program.

If ProvideX encounters a **RETURN** directive without an associated subroutine, ProvideX returns an Error #27: Unexpected or incorrect WEND, RETURN, or NEXT. **RETURN** must be the final directive in a compound statement.

Use the *RETURN label to emulate this directive in a statement reference.

Format 2: *Terminate Procedure, Specify Value*

RETURN *expression*[\$]

The **RETURN** directive serves a similar purpose in a multi-line function, embedded I/O, or OOP function procedure as it does in a typical subroutine. However, this version of a **RETURN** statement also passes back a value, *expression*[\$].

Multi-Line Function

When used in a multi-line definition of a user-defined function (**DEF FN**), the **RETURN** directive terminates the function procedure and passes back a result. The specified value must match the data type of the function definition itself.

Embedded I/O Procedure

The **RETURN** directive terminates an embedded I/O procedure and returns a value to the file handler, which it may use to determine the next operation. This is explained in greater detail in the *User's Guide*.

OOP Method Logic

In *Object Oriented Programming*, the **RETURN** directive terminates the method logic and passes back a result. See **Object Oriented Programming**, p.22.

See Also

GOSUB.. Execute Subroutine, p.141
SETESC Set Interrupt Processing, p.317
PERFORM Call Subprogram, Pass Variables, p.243
DEF FN Define Function, p.68
FUNCTION Declare Object Method, p.137
Called Procedures, User's Guide.

Examples

```

0100 ! 100
0110 INPUT "Enter start date DD/MM/YY: ",D$
0120 GOSUB 1000
0130 LET D1$=D$
0140 INPUT "Enter ending date DD/MM/YY: ",D$
0150 GOSUB 1000
0160 LET D2$=D$
0170 PRINT "DONE"; STOP
1000 ! 1000
1010 IF LEN(D$)<>8 THEN GOTO 1080
1020 IF D$(3,1)<>"/" OR D$(6,1)<>"/" THEN GOTO 1080
1030 LET D=NUM(D$(1,2),ERR=1080),M=NUM(D$(4,2),ERR=1080),Y=NUM(D$(7,2),ERR=10
1030:80)
1040 IF D<1 OR D>31 THEN GOTO 1080
1050 IF M<1 OR M>12 THEN GOTO 1080
1060 RETURN
1070 !
1080 PRINT "Invalid: Restart..."
1090 EXITTO 0100

0010 DEF FN%SHRINK$(LOCAL X$)
0020 LOCAL I
0030 I = POS(" "=X$)
0040 IF I <> 0 THEN X$=X$(1,I-1)+" "+X$(I+2); GOTO 0040
0050 RETURN X$
0060 END DEF

0010 DEF CLASS "Customer"
0020 PROPERTIES CUST_NO$,NAME$,ADDR1$,ADDR2$,CITY$,AMT_OWING, LAST_ORD_DT$
0030 FUNCTION Find(X$) Get_Customer
0040 FUNCTION Update() Upd_Customer
0050 LOCAL File OBJECT
0050 END DEF
0100 ON_CREATE: ENTER Db
0110 File = Db'Open("ARCUST")
0120 RETURN
0200 Get_Customer: ENTER C$
0210 READ (File'Channel,KEY=C$) ! Loads all the variables
0220 RETURN 1
0300 Upd_Customer:
0310 WRITE (File'Channel)
0320 RETURN 1

```

ROUND Directive

Control Rounding

Format **ROUND** {**ON** | **OFF**}

Description Use the **ROUND** directive to control the automatic rounding of values during calculations. If you use the **ROUND ON** directive, ProvideX will round all values to the **PRECISION** currently in effect before assigning them to numeric variables. The **ROUND OFF** directive terminates the rounding process. With **ROUND OFF**, ProvideX maintains all values at full 18-digit precision. The **ROUND** directive has no effect if **FLOATING POINT** mode is active.

Rounding Control

The unpredictable rounding of numeric values can cause problems during execution. It is very important to know how rounding works in ProvideX. If rounding is set to the default, a *divide* operation in ProvideX will maintain the precision of the starting value; e.g., $1014.475/100$ becomes 10.14475 which gets rounded to 10.145 . The automatic rounding of *intermediate* results can be turned off by setting the '**NR**' System Parameter, [p.676](#). Various other types of rounding can be controlled using the '**RN**'= System Parameter, [p.684](#).

See Also [PRECISION Change Current Precision, p.248](#),
[FLOATING POINT Switch to Scientific Notation, p.133](#)
['NR' System Parameter, p.676](#)
['RN'= System Parameter, p.684](#)
['RS' System Parameter, p.684](#)

Examples With **ROUND OFF**

```
0010 PRECISION 2
0020 ROUND OFF
0030 LET A=3*(2/3)
0040 PRINT A
-:run
2
```

With **ROUND ON**:

```
0010 PRECISION 2
0020 ROUND ON
0030 LET A=3*(2/3)
0040 PRINT A
-:run
2.01
```

RUN Directive *Transfer and Execute a Program*

Format `RUN [prog$[;entry$]][,ERR=stmtref]`

Where:

- `;entry$` Name of starting line label to use as entry point in the program. Optional. If included, append to the `prog$` string expression (e.g., `RUN "MY_PROG;STARTING_LABEL"`).
- `prog$` Name of the program to load and execute. Optional. String expression.
- `stmtref` Program line number or label to transfer control to.

Description Use the **RUN** directive to start or resume the execution of a program. You can include the name of a program to **LOAD** and then **RUN**. The **RUN** directive performs the following actions if a program is already loaded:

1. Resets the current precision to 2.
2. Enables rounding.
3. Clears **FOR/NEXT**, **GOSUB/RETURN**, and **WHILE/WEND** stack.
4. Resets **SETERR** and **SETESC** addresses and any saved **RETRY** address.

If you are loading a program or if the current program has not been interrupted, ProvideX begins execution at the program statement with the lowest line number. If the current program was interrupted (by an error, or a **BREAK** or **ESCAPE** directive) ProvideX resumes execution from the statement where the program left off.

You can embed the **RUN** directive in a program to provide an overlay facility. The **RUN** directive affects neither user data nor files. When you use it in a compound statement, the **RUN** directive must be the final directive.

You can assign an optional line label entry point for the **RUN** program. To do this, append a semicolon and the starting label name to the program name (e.g., `RUN "PROG;STARTING_LABEL"`). After the program is loaded, ProvideX internally issues a **GOTO** directive and starts execution at the assigned entry point.

If you set **'RR'**, ProvideX will also perform a reset for **RUN** directives.

See Also [CALL Transfer to Subprogram, p.40](#),
[PERFORM Call Subprogram, Pass Variables, p.243](#),
[RESET Reset Program State, p.288](#),
['RR' System Parameter, p.684](#)
[Saving, Loading, and Executing a Program, User's Guide.](#)

Example

```
-> RUN "INVGEN"
0040 RUN "PAY"+A$
```

SAVE Directive

Write Program to File

Format **SAVE** [**EDIT**] [*prog_name*\$[,*prog_size*][,**OWN**=*owner_id*[,**FLG**=*flg*:*flg*:*flg*]]]

Where:

- EDIT** Use the optional keyword **EDIT** with the **SAVE** directive to have ProvideX logically break the lines into segments and indent them. For more information refer to the '**LE**' System Parameter, [p.672](#) and the **LIST** Directive, [p.176](#).
- flg* Optional package flags. If you use these, delimit them with a colon ":" and use numeric expressions; i.e., integers between 1 and 25.
- prog_name*\$ Optional file name to receive the program. String expression. To save a program in a program library, use **[LIB]**, [p.781](#).
- prog_size* Program size. Optional. Numeric expression.
- owner_id* Optional package number (owner ID) to which this program is to be assigned. Numeric expression.



Note: **OWN=** and **FLG=** options are designed for use by registered application developers. These are used to encrypt and establish activation keys to run the programs.

Description

Use **SAVE** to copy/write the current program to a given *prog_name*\$. (Include the path name if the directory is neither current nor in your **PREFIX** definition.) If the file does not exist, it will be created. If *prog_size* is included, the file must not already exist.

If the output of the file is a program file type, ProvideX writes it to the file in internal (compiled) format. For serial/indexed file types, ProvideX writes the program in display (**LIST**) format. **SAVE** can only be used for serial, indexed, and program files.



Note: You can **RUN**, **CALL**, or **PERFORM** text files that contain programs just as you would any regular program file.

Structured Save. The **SAVE** directive can also be used to verify *modified* programs for structural integrity ('**SS**' System Parameter, [p.688](#)). Logical errors (e.g., a **FOR** with no corresponding **NEXT** or a **SWITCH** without an **END SWITCH**) will result in a Warning #125: Improper Structure Detected indicating where the fault was detected. For more information on logical structures, refer to the *User's Guide*, [Chapter 3](#).

See Also

SERIAL Create a Sequential File, [p.302](#)

Example

```
-->SAVE "PROG1A"
-->SAVE "/usr/a-r/PROGS/ARLIST"
```

You can **SAVE** or **SAVE EDIT** a text file containing the source of a program. Be sure to pre-create your **SERIAL** text file.

```
SERIAL "PROG00.TXT"
LOAD "PROG00"
SAVE EDIT "PROG00.TXT" ! Formatted with line breaks and indents
```

SAVE CONTROL Directive Save Image of Control

Format **SAVE CONTROL** *ctl_id* **TO** *filename*\$ [,**ERR=***stmtref*]

Where:

ctl_id Unique value of the control to capture, or 0 *zero* to capture the ProvideX desktop

filename A valid filename with the .bmp extension

stmtref Program line number or label to transfer control to.

TO Mandatory keyword, not case-sensitive.

Description Use the **SAVE CONTROL** directive to capture graphical controls or the ProvideX desktop and store the results in a bitmap (.bmp) file. These saved images can then be used by the **'PICTURE'** mnemonic. This command will not work on invisible objects/windows.

As most of the functionality is controlled through Windows API calls, it is possible to receive an Error #15: Operating system command failed if the operating system is unable to execute the necessary functions.



Note: In a WindX environment, the **SAVE CONTROL** directive is passed to the WindX client, therefore *filename* must be a file on the WindX client, not on the server.

See Also **'PICTURE'** Mnemonic, [p.631](#).

Example

```
00010 ! SAVE.CTL - Create a Chart then save Control & Screen images
00020 PRINT 'CS', 'FONT'("MS Sans Serif", -12), 'GF',
00030 !
00100 ! ^100 - Create the Chart
00110 Chrt=100; CHART Chrt, @(20,0,40,20), SEP=",", FMT="3DColumn", FNT="Verdana"
00120 CHART LOAD Chrt, "2,3,4,5,6,7,8,9,10,11,12,13/1,2,3,4,5,6,7,8,9,10,11,12/"
00130 Chrt 'Title1$="Title 1", Chrt 'Title2$="Title 2", Chrt 'Footer$="Footer"
00140 Chrt 'XAxisTitle$="X-Axis Title", Chrt 'YAxisTitle$="Y-Axis Title"
00150 Chrt 'CurrentSet=1, Chrt 'CurrentPoint=0, Chrt 'LegendText$="Item #1"
00160 Chrt 'CurrentSet=2, Chrt 'CurrentPoint=0, Chrt 'LegendText$="Item #2"
00170 Chrt 'PointText$="Jan,,Mar,,May,,Jul,,Sep,,Nov,,,"
00180 !
00200 ! ^100 - Save the images
00210 SAVE CONTROL Chrt TO "Chart.bmp", ERR=*NEXT
00220 SAVE CONTROL 0 TO "Screen.bmp"
00230 !
00300 ! ^100 - Print the saved images
00310 OPEN (1)*"winprt*;asis"
00320 PRINT (1)'PICTURE' (@X(0),@Y(0),@X(70),@Y(24), "Chart.bmp", 2),
00330 PRINT (1)'PICTURE' (@X(0),@Y(25),@X(70),@Y(49), "Screen.bmp", 2),
00340 CLOSE (1)
00350 END
```


SAVE DATA Directive

Save Program Constants

Format **SAVE DATA** *filename*\$ [,**ERR=***stmtref*], *varlist*

Where:

filename Name of Variable Definition file in which to store constant.

stmtref Program line number or label to transfer control to.

varlist List of variables.

Description This directive creates a Variable Definition file on disk that contains the names and values of variables named in *varlist*. These variables and their values can then be loaded into memory using the **LOAD DATA** directive. Variables loaded in this way are read-only (constants). Global variables are not supported.

To change the values of variables in a Variable Definition file, you must repeat the **SAVE DATA** process. When a Variable Definition file, is re-saved, the original contents are replaced.

See Also **LOAD DATA Load Program Constants, p.196.**

Example

```

CCOMPANY$="ABC Company"
DIVISION$="Laundry Division"
COMPANY_CODE=1
SAVE DATA "CO_DATA",COMPANY$,DIVISION$,COMPANY_CODE

START

LOAD DATA "CO_DATA"

DUMP
! ERR=0, CTL=0, RET=2
! Level=1
! PGN="<Unsaved>"
! Loaded data...CO_DATA (C:\Documents and Settings\Default
    User\Application Data\CO_DATA)
COMPANY$="ABC Company"
DIVISION$="Laundry Division"
COMPANY_CODE=1

COMPANY_CODE=2
Error #61: Authorization failure

```

SAVE FILE Directive

Save Bitmap to Disk

Format **SAVE FILE** (*chan*[,*ERR=stmtref*]) **TO** *filename*\$

Where:

chan Channel or logical file number to be read from.

filename A valid filename with a .bmp extension.

stmtref Program line number or label to transfer control to.

TO Mandatory keyword, not case-sensitive.

Description Use the **SAVE FILE** directive to save an image written to ***BITMAP*** (virtual file) directly to a bitmap file (.bmp) file on disk.



Note: This capability is for Windows only. The bitmap *chan* can be a WindX-connected file if the pathname contains a "[WDX]" prefix. Also, if the pathname contains "[WDX]" then the *chan* must be a WindX file.

See Also ***BITMAP* Virtual Bitmap**, *p. 738*

Example In the code below, ***BITMAP*** is used to capture ProvideX internal bitmaps, which are then saved to .bmp files using **SAVE FILE**:

```
0010 PicList$='picture'(*)+", "
0020 x=pos(", "=PicList$); if x=0 then stop
0030 f$=PicList$(1,x-1),PicList$=PicList$(x+1)
0040 open (1)"*bitmap*;Width=1;Length=1"
0050 print (1)'picture'(0,0,@x(mxc(1)+1),@y(mxl(1)+1),"!"+f$,4),
0060 f$="/tmp/"+f$+".bmp"; erase f$,err=*proceed
0070 save file (1) to f$
0080 close (1)
0090 goto 0020
```

SELECT..FROM..NEXT RECORD Directive Query Records

Format

1. Open, read and query records: **SELECT** *iolist* [,REC=*name*\$] **FROM** {*filename*\$ | *chan*} [,KNO=*num* | *name*\$] [**BEGIN** *key*\$ **END** *key*\$] [,ERR=*stmtref*] [**WHERE** *expression*] ..**NEXT RECORD**
2. Return full record contents: **SELECT RECORD** *iolist* **FROM** {*filename*\$ | *chan*} [,KNO=*num* | *name*\$] [**BEGIN** *key*\$ **END** *key*\$] [,ERR=*stmtref*] [**WHERE** *expression*] ..**NEXT RECORD**
3. Return key portion of the record: **SELECT KEY** *iolist* [,REC=*name*\$] **FROM** {*filename*\$ | *chan*} [,KNO=*num* | *name*\$] [**BEGIN** *key*\$ **END** *key*\$] [,ERR=*stmtref*] [**WHERE** *expression*] ..**NEXT RECORD**

Where:

BEGIN <i>key</i> \$	Keyword and string expression used to establish the begin and end keys for the selected range. <i>key</i> \$ may also be a colon-separated list of values for use with segmented key definitions (as per KEY= in the READ Directive, p.272).
END <i>key</i> \$	
<i>chan</i>	Channel or logical file number to be read from.
<i>expression</i>	Condition must return <i>true</i> or <i>false</i> . Numeric or string expression.
<i>filename</i> \$	Name of the file to be opened and read from. String expression.
FROM	Keyword required to indicate <i>filename</i> \$ or <i>chan</i> .
<i>iolist</i>	List of variables to be read from the file. The order in which the variables are specified (<i>A</i> \$, <i>B</i> \$, <i>C</i> \$, ..) corresponds to how the fields are read from each record (1st, 2nd, 3rd, ..). If you use an * <i>asterisk</i> then all fields defined by the embedded data dictionary will be returned. You can use an IOL= <i>iolref</i> as your <i>iolist</i> .
KEY	Optional keyword for specifying that the value of the key is to be treated as if it were the data.
KNO= <i>num</i> <i>name</i> \$	File access key value (<i>num</i>) or name (<i>name</i> \$).
NEXT RECORD	Directive required to end the SELECT loop.
REC= <i>name</i> \$	Record prefix (REC=VIS (<i>string</i> \$) can also be used).
RECORD	Optional keyword for returning the record contents in a single variable.
<i>stmtref</i>	Program line number or label to transfer control to.

Description

Use the **SELECT** directive to open, read and query records **FROM** the specified data file, or just to read data from a specified file number. As each record is read, ProvideX processes any logic you include following the **SELECT** directive up to the **NEXT RECORD**. When ProvideX encounters a **NEXT RECORD** statement with no records found for a nested **SELECT**, it will locate the corresponding **SELECT** statement.

If you include a **WHERE** clause, ProvideX will process only those records **WHERE** the condition is *true*.

The use of **BEGIN** and **END** will restrict the range of records read. These clauses are only supported for Keyed and Memory files. They can be used with WindX-connected files. Note that, if you are using **BEGIN** and **END** in **SELECT** statements for files with descending keys, the **END** value must be lower than the **BEGIN** value.

If a **SELECT** directive specifies a *filename\$*, that file will be opened when the **SELECT** is executed, thus setting the file pointer to the beginning of the file. If the directive uses a channel, **SELECT** will begin from the current file pointer. When using the latter, it may be prudent to include a **BEGIN** clause to reset the file pointer; e.g., `SELECT * FROM 1 BEGIN ""`.



Note: Every **SELECT** must have a corresponding **NEXT RECORD** directive, and must be in the correct sequence. A mis-matched number of **SELECT** and **NEXT RECORD** directives can result in either an Error #27: Unexpected or incorrect WEND, RETURN, or NEXT, or an Error #28: No corresponding FOR for NEXT.

Because ProvideX pads descending keys to their full length with `$$$`, the **BEGIN** value is *key+\$FF\$* and the **END** value should be *key+\$00\$* so that the ending key is less than the beginning key; i.e., the correct format is currently `SELECT * FROM filename BEGIN key$ END key+00`.

Although the incorrect statement `SELECT * FROM filename BEGIN key$ END key+FF` may have worked in prior versions, it no longer does as of Version 4.20.

You must either include a **NEXT RECORD** directive to end the **SELECT** loop or instruct ProvideX to exit the loop early (with an **EXITTO** directive). When an **EXITTO** directive is used, the file will be closed if **SELECT** specified a data file name rather than a channel.

Also, in earlier versions of ProvideX, the **CONTINUE** and **BREAK** directives (and corresponding `*CONTINUE` and `*BREAK` labels) were not supported for use with **SELECT .. NEXT RECORD** directives. It is now possible to include **BREAK** and **CONTINUE** commands in **SELECT** structures.



Note: When `SELECT * FROM filename BEGIN key$ END key+FE` is encountered when processing a MySQL table, the SQL sent to the server will be optimized to reduce records processed by the server.

Formats 2 and 3: **SELECT RECORD** and **SELECT KEY Options**

The **SELECT .. NEXT RECORD** statement can include syntax for full record contents or key portion of the record. **SELECT RECORD** allows the specification of a single variable, or `*` as per the **READ RECORD** directive. **SELECT KEY** reads the file and treats the key contents as if it were the data. A single variable can be specified to receive the key value or a formatted IOList; e.g., `SELECT KEY DIV: [CHR(2)] , CLIENT: [CHR(7)] FROM "ABC"`. Refer to the examples below.

Examples

The following illustrates use of the **SELECT WHERE** clause:

```

0010 SELECT IOL=0100 FROM "VEND_FILE",KNO=1 BEGIN "ABC CO." END "THAT CO." WHERE
0010:CITY$="CLARENDON"
0020 PRINT REC(IOL=0100)
0030 NEXT RECORD
0100 IOLIST VEND$,NAME$,ADDR1$,ADDR2$,CITY$,PROV$,POSTAL$,INV_DT$,INV,
0100:AMT,TERMS,DUE_DT$
0110 PRINT "DONE"; END

```

In the following example, **SELECT KEY** is issued to return the key portion of the record:

```

SELECT KEY cst_name$,REC=X1$ FROM "cstfile",KNO=1 BEGIN StartName$ END
      StartName$+$$FF$ WHERE x1.cst_name$<>SkipName$
PRINT x1.cst_name$
NEXT RECORD

```

This uses the **SELECT RECORD** directive to return the full record contents:

```

SELECT RECORD r$ FROM "cstfile",KNO=1 BEGIN "D" END "D"+$7F$
PRINT r$
NEXT RECORD

```

See Also

BREAK Immediate Exit of Loop, *p.33*,
CONTINUE Initiates Next Iteration of Loop, *p.57*,
EXITTO End Loop, Transfer Control, *p.125*,
READ RECORD Read Record from File, *p.275*.

SERIAL Directive

Create a Sequential File

Format **SERIAL** *filename* \$[,*max_recs* [,*rec_size*]] [,**ERR=***stmtref*]

Where:

- filename*\$ String variable that defines the name of the serial (sequential) file to create.
- rec_size* Maximum size of the data portion of the record. (Optional on most operating systems.) Numeric expression.
- max_recs* Estimated number of records the file is to contain. The default is no initial allocation of file space, with no limit as to final size. (Not used in most operating system implementations.) Numeric expression.
- stmtref* Program line number or label to transfer control to.

Description When you use the **SERIAL** directive to create a serial (sequential or flat) file, ProvideX creates a standard text data file in a format the operating system can access directly. The record size is "for documentation purposes only" on most operating systems. If you do specify the size, make it large enough to hold all the data fields written to the file for each record. If a file of the name you use already exists, ProvideX returns an Error #12: File does not exist (or already exists).

Record Access Mode and Binary Access Mode

In *record access mode*, a serial file is read a line or record at a time. Each line is determined by the presence of end-of-line-character(s) which are different based on the OS. On UNIX (or similar OS's) the end-of-line character is the line feed (\$0A\$), on Microsoft Windows it is a carriage return followed by a line feed (\$0D0A\$).

Each **READ** or **READ RECORD** will only read 1 line. When using a standard **READ** directive **READ** each line will be parsed by any field SEP's that exist within the line and the corresponding variables within the **READ** will be set. If you use a **READ RECORD** then the field SEP's will not be parsed and the variable on the directive will receive the complete lines data.

When you **OPEN** a serial file in record access mode, there are 2 logical file pointers in the file. The *read pointer* starts at the top of the file, and the *write pointer* starts at the bottom; therefore, reads return the data starting from the first record whereas writes automatically append to the end of the file.

You can move both file pointers by issuing a **READ** or **WRITE** and supplying an **IND=** clause (IND=0 is the first record, IND=1 is second, etc.)

When a file is opened in *binary access mode*, you can read and write byte by byte with no regard for the data contents. The data on the file is logically parsed into records each of whose size is based on the **ISZ=** in an **OPEN** clause. For example,

`OPEN(chan , ISZ=1)` sets binary access mode with the contents of the file being considered a series of 1-byte records. An `OPEN(chan , ISZ=512)` sets binary mode with a series of 512-byte records.

A **READ** or **READ RECORD** will read the **ISZ=** number of bytes and treat these as a record. Using an **IND=** moves the current file pointer to a specific record, so when using **ISZ=1**, the **IND=** will take you to a specific byte offset within the file.

WindX supports the use of this directive via the `[WDX]` tag; e.g., `SERIAL "[WDX]somefile.ext" . . .` For more information, see [\[WDX\] Direct Action to Client Machine, p.801](#).

Examples

Both examples below create files whose structures are viewed by ProvideX as serial (or unknown type):

```
0010 SERIAL "PRNTFL" , , 133
0010 SERIAL A$+ "-" +B$, 100 , 50 , ERR=1090
```

See Also

[Labels/Logical Statement References, p.816](#)
['NN' System Parameter, p.676](#),
[LOAD Read Program into Memory, p.194](#),
[SAVE Write Program to File, p.295](#).
[File Types, User's Guide](#)

SET_FOCUS Directive

Set Input Focus

Formats

1. *Set Focus On*: `SET_FOCUS ctl_id [,ERR=stmtref]`
2. *Retry*: `SET_FOCUS RETRY ctl_id [,ERR=stmtref]`
3. *Read CTL Value*: `SET_FOCUS READ ctl_id [,ERR=stmtref]`

Where:

ctl_id Unique numeric control ID of the custom control (**BUTTON**, **LIST_BOX**, **DROP_BOX**, etc.) to which to direct subsequent input.

stmtref Program line number or label to transfer control to.

Description

Use the **SET_FOCUS** directive to set the focus on the custom control you want to receive the next user input. ProvideX returns Error #65: Window element does not exist or already exists if you use **SET_FOCUS** for a non-existent control item. An Error #13: File access mode invalid is returned if the control is disabled.

Format 1: Set Focus On

SET_FOCUS *ctl_id*

Setting the focus to a control directs subsequent terminal input to it, unless the user overrides it with the mouse. If the value of the *ctl_id* is zero, input is directed to the screen **INPUT** directives.

Example:

```
1000 BUTTON 100,@(10,10,10,2)="Write"
1010 BUTTON 101,@(15,10,10,2)="Delete"
1030 SET_FOCUS 100
```

The next keyboard input will be directed to the *Write* button.

Format 2: Retry

SET_FOCUS RETRY *ctl_id*

Use the **SET_FOCUS RETRY** format to set an internal "modified" flag on the control. If the user attempts to **Tab** away from the control, it will retry its associated CTL event. You can use this in input validation (i.e., when an application detects invalid input). You can issue a **SET_FOCUS RETRY** directive to have ProvideX reprocess the input field whether the user changes something or attempts to **Tab** away from it.

Format 3: Read CTL Value

SET_FOCUS READ *ctl_id*

Use **SET_FOCUS READ** to obtain the CTL value of the control which currently has focus. The CTL value is 0 *zero* if no control has focus.

SET_NBF Directive *Set Number of Keyed I/O Buffers*

Format **SET_NBF** *buffers*

Where:

buffers Number of internal key I/O buffers. Numeric expression.

Description *Obsolete.*



Note: This directive is included here for completeness only. **SET_NBF** is obsolete and has been replaced by the **'BF'= System Parameter, p.657**.

Use the **SET_NBF** directive to adjust the number of internal buffers to be used by the key I/O manager. Use values in the range 0 to 25. By allocating more buffers, you can typically obtain better Keyed access performance. Normally the default of 10 buffers yields good overall performance. If you use the **'BF'** parameter to set the number of buffers to zero, this will force ProvideX to allocate local or file-specific buffers (controlled by the parameter **'FB'**).

Using this directive to optimize system performance takes a great deal of experience. Many factors influence the number of buffers required. The system may slow down if too many buffers are assigned, causing excessive queue searching. Too few buffers may cause additional disk I/O's. In a single user environment, more buffers will typically improve performance, since ProvideX will use these buffers to cache disk reads and writes.

SET_PARAM Directive

Set System Parameters

Format **SET_PARAM** [*param_list*]

Where:

param_list List of mnemonics and optional values for setting various system parameters for your ProvideX environment.

Description Use the **SET_PARAM** directive to set system parameters. These parameters set internal options in ProvideX. For the list of possible parameters that can be set, see [Chapter 6. System Parameters, p.653](#).

You can find the current value or state of these parameters using the **PRM()** function and the system variable **PRM**.

See Also [PRM System Variable, p.570](#)
[PRM\(\) Function, p.504](#).

Examples

```
0100 PRINT PGN
0110 SET_PARAM 'OP'
0120 PRINT PGN
-:run
C:\Program Files\Sage Software\ProvideX\MANUAL\TST\TST_EGS
TST_EGS

SET_PARAM 'AH' ! Switches Alternative Heading ON
SET_PARAM -'AH' ! Minus sign switches 'AH' OFF
SET_PARAM 'BY'=0 ! sets the Base Year to Julian calendar base
SET_PARAM 'BY'=1970 ! resets Base Year to default

You can use a Boolean value to reset a switch; e.g.,

SET_PARAM 'AH'=0 ! switches 'AH' off
->?prm('ah')
0
```

SETCTL Directive

GOSUB on CTL Event

Format **SETCTL** *ctl_id:stmtref*

Where:

ctl_id CTL value to intercept. Numeric expression.

stmtref Program line number or label to transfer control to.

Description When ProvideX executes the **SETCTL** directive, it intercepts CTL values on **INPUT** statements and transfers control via a **GOSUB** to the specified line number or label. On completion of the subroutine (e.g., on a **RETURN**) control will pass back to the **INPUT** statement where the event was intercepted.

To reset a **SETCTL**, set *stmtref* to 0000. If *stmtref* = 0000, the logic to intercept the CTL value is deleted.



Note: **SETCTL** is only valid for the program it was executed in, not any subsequent programs (e.g., those initiated through **CALL**, **PERFORM** or **RUN** directives).

See Also [GOSUB.. Execute Subroutine, p.141](#),
[CTL System Variable, p.557](#).

Example

```
0010 SETCTL 3:2000
0020 SETCTL 4:9000
0100 INPUT (0,err=0100) @(x,y), "Enter Name:",A$
0110 STOP
2000 ! refresh screen
2010 PRINT 'RS',; RETURN
9000 ! exit routine
9010 EXITTO 9900
9900 END
```

In the above example, whenever the CTL=3 on the **INPUT** statement (i.e., the user hits the **F3** key), control is passed to the subroutine at line 2000. When the **RETURN** statement is executed, control passes back to the line that contained the **INPUT** statement.

SETDAY Directive

Change Local Date

Format **SETDAY** [date\$]

Where:

date\$ Date to assign to the **DAY** variable for the current session. String expression.

Description Use the **SETDAY** directive to set or change the current local processing date for the current user and session (returned in the **DAY** variable). The **DAY_FORMAT** of the string-expression is the default, MM/DD/YY, unless you specify a different setting in a **DAY_FORMAT** directive. ProvideX will continue to update the altered date based on the current time and date.

The altered date will stay in effect until the end of the session or until the execution of a **START** directive. Then ProvideX reverts to the operating system's current date. Note that this directive calculates an offset from the current operating system date/time. If the operating system date is altered after the execution of the **SETDAY** directive, a corresponding change will be reflected in the value of **DAY**.



Note: This directive only affects the user executing the directive.

See Also [DAY System Variable, p.557](#)
[DTE\(\) Function, p.422](#)

Example

```
0090 SETDAY "11/15/00"
0090 PRINT DAY
0110 SETDAY "02/26/00"
0120 PRINT "Today's date is ",DAY
->RUN
11/15/00
Today's date is 02/26/00
```

The date 02/26/00 is only in effect for the current session. If the operating system's date and time are advanced by 2 days (to the 28th of November) during the current session, then the current session date 02/26/00 will be advanced to 02/29/00 for the current user.

SETDEV Directive

Set Device Type Name

Format **SETDEV** (*chan*)/*lcs_devtype*\$

Where:

chan Channel or logical file number of the file for which the device type is being overridden.

lcs_devtype\$ The lower-case string value defining the device type. Its maximum length is 12 characters. String expression.

Description Use the **SETDEV** directive to make dynamic changes in the value of the **DEVICE TYPE** field maintained by ProvideX for device files. This directive is applicable primarily to device drivers, and then only those that can support more than one device type on the same port.

When you use this directive, the device driver **"*DEV/DIAL-UP"** changes the device type field for a dial-up terminal based on the type of terminal actually connecting to ProvideX.

SETDEV IOL= Directive *Alter IOList for Open Channel*

Format **SETDEV** (*chan*) **IOL=** *iolref*\$ [: { * | ^ | **KEY** }]

Where:

- * Optional indicator, changes the embedded IOList.
- ^ Optional indicator for changing the alternate IOList
- chan* Channel or logical file number.
- iolref*\$ Valid IOList reference.
- KEY** Optional indicator for changing the KEY IOList



Note: If no { * | ^ | **KEY** } indicator is used, the default IOList will be changed.

Description Use the **SETDEV IOL=** directive to alter the various IOLists associated with a currently open channel. This will override the current IOList definition for a channel while the channel is open, and will not physically alter the file.

Example

```

0020 IOLIST A$,B$,C$
0030 SETDEV (1)IOL=0020:* ! changes the embedded IOList
0040 SETDEV (1)IOL=0020:^ ! changes the alternate IOList
0050 SETDEV (1)IOL=0020:key ! changes the KEY IOList
0060 X$cpl("IOLIST A$,B$")
0070 SETDEV (1)IOL=X$ ! changes the default IOList

```

SETDEV KEY Directive

Alter Keys of Open Channel

SETDEV (*chan*) **KEY:** *numexpr*,*strexpr*

Where:

chan Channel or logical file number.

numexpr Valid KNO value (file access key) for the file.

strexpr New key name.

Description Use the **SETDEV KEY** directive to either assign or alter named keys for an open channel. The name assigned to any given key is only valid while the channel is open, and will not physically alter the file.

Example

```
->OPEN (1) "*msglib.en"  
->SETDEV (1)KEY:0, "PrimaryKey"  
->PRINT RCD(1,KEY="!PRINT",KNO="PrimaryKey")  
{&P!Print}
```

SETDEV PROGRAM Directive

Set I/O Program

SETDEV (*chan*) **PROGRAM** "*prog_name*"

Where:

chan Channel or logical file number.

prog_name Name of Embedded I/O program.

Description The **SETDEV PROGRAM** directive allows an Embedded I/O program to be attached to an open channel if the file does not already have one associated with it. The embedded I/O program will remain in affect only until the channel is closed.

Example

```

->load"Embedded.io"
->list
0010 ! Embedded.io
0020 Pre_Close:
0030 PRINT "This message generated by the Pre_Close routine"

->OPEN(1)*msglib.en      ! Open a file
->SETDEV (1) PROGRAM "Embedded.io" ! Attach EIO Program
->CLOSE (1)      ! Close file
This message generated by the Pre_Close routine
->

```


SETDEV SEP= Directive

Change File SEP

SETDEV (*chan*) **SEP=***char*\$

Where:

chan Channel or logical file number.

char\$ Field separator character. Hex or ASCII string value.

Description

The **SETDEV SEP=** directive allows an application to change the standard field separator character on a file-by-file basis. If the value of *char*\$ is longer than 1 character, the system will generate a string length error. If the value of *char*\$ is null, the file will be set to have dynamic separators (length delimited).



Note: This only affects the currently opened channel and does not physically alter the field separator value stored within the file header – it reverts back to its original value when the file is subsequently opened. This is applicable to native ProvideX files only.

SETDEV TSK() Directive

Add to TSK() List

Format **SETDEV TSK()** *task\$*

Where:

task\$ String of characters to add to the **TSK()** table. String expression.

Description Use the **SETDEV TSK()** directive to add a string to the end of the internal table for the **TSK()** function. Each time **SETDEV TSK()** is used, the *task\$* is evaluated and appended to the internal table.

Do not include a task number in the **SETDEV TSK()** directive. ProvideX automatically supplies the next available number in sequence. The order of the table is determined by the order in which **SETDEV TSK()** directives are executed. ProvideX places the string from the first **SETDEV TSK()** into **TSK(0)**, the second into **TSK(1)** and so on. If you use the **TSK()** function with an invalid task number, ProvideX returns an Error #41: Invalid integer encountered (range error or non-integer).

See Also [TSK\(\) Function, p.543](#).

Example

```
0100 SETDEV TSK("lp - hp laserjet on johns desk"
0110 SETDEV TSK("p1 - epson printer - spooled"
0120 PRINT TSK(0), 'LF', TSK(1)
LP - hp laserjet on johns desk
P1 - epson printer - spooled
```

SETDRIVE Directive

Change Default Drive

Format **SETDRIVE** {*letter*\$ | *number*}

Where:

letter\$ String expression. Drive letter from A to Z representing the new disk from which you want to work or run applications.

number Numeric expression. Number from 0 to 25 representing the drive letter (A to Z) from which you want to work or run applications.



Note: This directive only functions with PVX for Windows, and is used primarily in the conversion from legacy DOS-based Business BASIC languages.

Description Use the **SETDRIVE** directive to change the default disk drive.

See Also **CWDIR** [Change Working Directory, p.62.](#)

SETERR Directive

Set Error Transfer

Formats

1. *Error Transfer to Line/Label*: **SETERR** *stmtref*
2. *Error Transfer to Program*: **SETERR** *prog\$*[:*entry\$*]

Where:

- ;entry\$* Optional entry label in the error-trapping program. Define once per session.
- prog\$* Name of a generic error-trapping program. Define it once per session.
- stmtref* Program line number or label to transfer control to.

Description

Use **SETERR** to define the transfer address for any error for which you have not used the **ERR=** option. While a **SETERR** is in effect, any error (divide check, subscript range error, etc.) transfers control to the statement number, label, or program specified in **SETERR**. To disable the **SETERR** transfer in a program, use statement number 0000 as *stmtref*.

Once a **SETERR** transfer occurs, ProvideX inhibits further **SETERR** transfers until either another **SETERR** is executed or a **RETRY** directive re-executes the statement which caused the error. This prevents system looping caused by errors in an error handler.

The chart below shows the order of precedence for error handling:

ERR=	Error trapping at the line level
SETERR <i>stmtref</i>	General error trapping within a program
SETERR " <i>prog;entry</i> "	General error trapping for the current session. Entry point is optional.
<i>... else ...</i>	Drop to the console and report the error (provided the ' XT ' parameter is disabled).

ERROR_HANDLER READ can be used to determine the current **ERROR_HANDLER** or **SETERR** program in effect.

See Also

[ERROR_HANDLER Define Generic Handler, p. 121](#)
Error Processing in the *ProvideX User's Guide*

Example

```

3160 SETERR 3230
3170 DATA 1,2,3,"CAT"
3180 DATA 4,5,6,"DOG"
3190 READ DATA IOL=3220
3200 PRINT IOL=3220
3210 GOTO 3190
3220 IOLIST X,Y,Z,A$
3230 PRINT "GOTCHA"
3240 SETERR 0000 ! Disables SETERR
3250 STOP
--:GOTO 3160; BEGIN; RUN
1 2 3CAT
4 5 6DOG
GOTCHA

```

SETESC Directive

Set Interrupt Processing

Formats

1. *Subroutine Interrupt-Handler*: **SETESC** *stmtref*
2. *Subprogram Interrupt-Handler*: **SETESC** *prog_name\$*
3. *Interrupt Process On/Off*: **SETESC** {**ON** | **OFF**}
4. *Enable/Disable for Range*: **SETESC** {**DISABLE** | **ENABLE**}

Where:

prog_name\$ Name of generic interrupt-handling program. Define it once per session.

stmtref Program line number or label to transfer control to.

Description

Use the **SETESC** directive to

- Specify the subroutine or subprogram to handle a break request.
- *Enable/disable* recognition of the **Break** or **Esc** keys in ProvideX.

Use this directive to prevent a user from breaking out of a program during critical periods or for security reasons.

Format 1: *Subroutine Interrupt-Handler*

SETESC *stmtref*

Use this format to specify a subroutine written to handle break requests. If a **SETESC** is active and the user enters a **Break** or **Esc**, ProvideX performs a **GOSUB** to the statement you designate in this format of the **SETESC** directive. Use the **RETURN** directive if you want control to return to the original program flow after the subroutine processes the break request. To deactivate the **SETESC** directive, set the *stmtref* to 0000.

Example:

```
0010 SETESC BREAK_IT
0020 FOR I = 1 TO 100000
0030 PRINT I
0040 NEXT I
0050 STOP
1000 REM -- BREAK HANDLER ---
1010 BREAK_IT:
1010 PRINT "Please wait till I'm finished"
1030 PRINT "... I've lost track"
1040 PRINT " I'll have to start again"
1050 I=1
1055 WAIT 5 ! Or messages flash by, user doesn't see, breaks again...and again
1060 RETURN
```

To terminate **SETESC** handling:

```
0020 SETESC 0000
```

Format 2: Subprogram Interrupt-Handler**SETESC prog_name\$**

Use this format to specify a program ProvideX is to **CALL** automatically to process **BREAK/ESCAPE** signals. When the program exits, execution resumes.

Format 3: Interrupt On/Off in Program**SETESC {ON | OFF}**

If you use **SETESC ON**, ProvideX recognizes the user's **Ctrl-Break** input and halts execution. If you apply a **SETESC OFF** directive in your current session or program, ProvideX treats the user's subsequent use of the break keys as equivalent to a carriage return or **Enter**. That is, the user's **Esc** key or **Ctrl-C** terminates the current input (e.g., by advancing to the next input) but does not let the user halt a program in Execution mode from the keyboard. A **Ctrl-Break** retries the current input but does not halt program execution.

If you use **SETESC ON**, ProvideX recognizes the user's **Ctrl-Break** input and halts execution.

Example:

```
0010 SETESC BREAK_IT
0020 DATA 1,2,3,"CAT"
0030 DATA 4,5,6,"DOG"
0040 DATA 7,8,9,"PIG"
0050 DATA 3,2,1,"DONE"
0060 READ DATA A,B,C,X$,ERR=0110
0070 PRINT A,B,C,X$
0080 INPUT "Try to break out: ",Y$
0090 GOTO 0060
0100 BREAK_IT: PRINT "That was easy."; STOP
0110 PRINT "Error on the READ (EOF)"
0120 STOP
```

Results when run under Windows with **SETESC ON**, then **SETESC OFF**:

User Enters:	--:SETESC ON --:RUN 1 2 3CAT	--:SETESC OFF --:RUN 1 2 3CAT
Enter	Try to break out: 4 5 6DOG	Try to break out: 4 5 6DOG
Esc	Try to break out: 7 8 9PIG	Try to break out: 7 8 9PIG

User Enters:	-:SETESC ON -:RUN 1 2 3CAT	-:SETESC OFF -:RUN 1 2 3CAT
Ctrl-C	Try to break out: 3 2 1DONE	Try to break out: 3 2 1DONE
Ctrl-Break	Try to break out: That was easy. -:	Try to break out: Try to break out: Try to break out: Try to break out: Try to break out:

The **SETESC {ON | OFF}** directive is in effect for the duration of the session or until you reverse it with another **SETESC {ON | OFF}**.

Format 4: *Enable/Disable for Range*

SETESC {DISABLE | ENABLE}

When you use the **SETESC DISABLE** directive, that disables **SETESC** for a range of statements. ProvideX ignores all subsequent **SETESC *stmtrefs*** until a **SETESC ENABLE** is encountered. You can make use of the **DISABLE** option in debugging a program which has embedded **SETESC** directives.

SETFID Directive

Set FID() Definition

Format **SETFID [(chan)] fid_def\$**

Where:

chan Optional channel or logical file number.

fid_def\$ String value to define **FID(0)**. Its maximum length is 12 characters.
String expression.

Description Use the **SETFID** directive to make dynamic changes in the value returned by the **FID()** function. Any open channel may have its FID value changed. If *chan* is not provided, it defaults to affect channel 0. This is most commonly used to change the FID value of channel 0 (i.e., the terminal) for legacy applications that require unique FID values.

By default, ProvideX uses the value of the operating system environment variable **PVXFID0** as the value for **FID(0)**. If this environment variable is not defined, ProvideX will dynamically assign a value starting at T0 (T zero).

See Also [FID\(\) Function, p.438.](#)

Example In this example, the value in the **WHO** system variable is something like "SMITHJ":

```
0010 ! START_UP
0020 OPEN (1)"MYCONFIG"
0030 READ (1,KEY=WHO,ERR=0050)X$
0040 SETFID X$
0050 CLOSE (1)
```

The following example defines a unique session and terminal **FID** in a Windows environment, given a directory called C:\Program Files\Application\FIDS with the files "T5", "T5", "T6", "T7", "T8", and so on. (This could also be done with terminal IDs of 3 characters or more.)

In a **START_UP** program,

```
0010 OPEN (1) "C:\Program Files\Application\FIDS"
0020 READ (1,END=1000) ID$
0030 IF ID$(1,1)="." GOTO 0020 ! Skip "." and ".."
0040 %FID_FILE=GFN
0050 OPEN LOCK (%FID_FILE,ERR=0020) PTH(1)+DLM+ID$
0060 SETFID ID$
0070 CLOSE (1)
....
1000 MSGBOX "Sorry. No free FID for this station. End another session & try again"
1010 QUIT
```

This logic will open and leave open a file pertaining to the station FID. Once the task ends, the file will be available for another session.

SETMOUSE Directive

Control/Set Mouse

Formats

1. *Define Mouse Region*: **SETMOUSE** [*]@(*col,ln,wth,ht*){= | :}[*expression*]
2. *Use String to Define Event*: **SETMOUSE** [*]*string*#{= | :}[*expression*]
3. *Use Current Window Size*: **SETMOUSE** [*]@(*){= | :}[*expression*]
4. *Clear All Settings*: **SETMOUSE CLEAR**
5. *Enable/Disable*: **SETMOUSE** {ON | OFF}

Where

- * Optional. The first instance of an asterisk in Formats 2 through 4 above defines the mouse event as being for all windows. The second instance in Format 4 is a size option. See the next row of this chart.
- @(*col,ln,wth,ht*) Mouse region coordinates. Numeric expressions: starting column, starting line, width (number of columns) and height (number of lines). Using an optional single *asterisk* * instead of *col,ln,wth,ht*, defines the mouse region for the event as equal in size to the current window.
- = | : Operators control mouse event processing:
 - : (colon) when mouse button is pressed/dragged in the region.
 - = (equals sign) when mouse button is released in the region.
- expression* Expression to define the type of function for mouse events in the specified region: if *numeric*, the CTL value to return; if *string*, the directive to execute.
- string*\$ Character string to be compared to the current screen contents. String expression.

Description

Use the **SETMOUSE** directive to define and control mouse events. By default, the mouse events are tied to the current window only. The **SETMOUSE** directive now supports fractional coordinates to two decimal places.



Note: Only a single **SETMOUSE** directive can be active for the same region at any time.

Format 1: Define Mouse Region

SETMOUSE [*]@(*col,ln,wth,ht*){= | :}[*expression*]

Use this format to define the region in which a mouse event can occur. Use an optional *asterisk* * to define the mouse event as being for all windows instead of just the current window.

Example:

```

0010 REM Return CTL=4 when mouse released on line 20 in columns 0 through 5
0010 SETMOUSE ON
0020 PRINT 'CS',@(0,20),"[Quit]",
0030 SETMOUSE @(0,20,6,1)=4
0100 INPUT @(10,10),"Enter name:",X$
0110 IF CTL=4 THEN GOTO 9000 ! Wrap-up

```

Format 2: Use String to Define Event

SETMOUSE [*]string\$ { = | : } [expression]

Use this format to define a character string to set the function the mouse event will generate. To remove a mouse event definition use a null " " expression. Use an optional *asterisk* * to define the mouse event as being for all windows instead of just the current window.

Example:

This logic returns CTL=6 whenever the mouse is released on the string 'Help' and CTL=4 on the string 'Quit'. Whenever the mouse is released on the word 'Calc', ProvideX calls "CALC" then returns to the current statement after "CALC" is executed.

```

0010 SETMOUSE ON
0020 SETMOUSE *"Help"=6 ! Help request
0030 SETMOUSE *"Quit"=4 ! Quit
0040 SETMOUSE *"Calc"="CALL " "CALC" " "

```

Format 3: Use Current Window Size

SETMOUSE [*]@(*) { = | : } [expression]

The first optional *asterisk* * defines the mouse event as being for all windows instead of just the current window. Use the second *asterisk* * in this format to define the mouse region as equal to the dimensions of the current window.

Format 4: Clear All Settings:

SETMOUSE CLEAR

Use this format to remove all previously defined **SETMOUSE** regions.

Format 5: Enable/Disable Mouse (DOS Only)

SETMOUSE {ON | OFF}

This format is included for completeness only. It applies to legacy (character-based) systems running under DOS.

SETTIME Directive

Set Local Time

Format **SETTIME** [*time*]

Where:

time Current time of day for the user session. Numeric expression.

Description Use the **SETTIME** directive to set or change the current processing time (returned in the **TIM**, **TME**, and **TMS** variables). The value of *time* must be between 0 and 24 (the desired time of day based on a 24 hour clock). Once set, the time will be continuously updated based on the operating system clock.

ProvideX will continuously update the date and time based on the set time until the end of the session or until the execution of a **START** directive. Then, ProvideX reverts to the operating system's time.

Note that this directive calculates an offset from the current operating system date/time. If that date/time is altered after the execution of the **SETTIME** directive, a corresponding change will be reflected in the values of **TIM/TME/TMS**.



Note: This directive only affects the user executing the directive.

See Also

[TIM System Variable, p.573](#),
[TME System Variable, p.574](#),
[TME System Variable, p.574](#),
[DTE\(\) Function, p.422](#).

Example

```
0100 PRINT TIM
0110 SETTIME 1.10
0120 PRINT "Current time",TIM
RUN
8.51
Current time 1.10
```

If the OS time were advanced by an hour (i.e., from 8.51 to 9.51), that hour would also be added to the time for the current session, advancing the current session's time to 2.10 and setting **TIM**, **TME** and **TMS** to the new value.

SETTRACE Directive

Enable Program Tracing

Formats

1. Set Trace, Define File: **SETTRACE** [(*chan*)]
2. Trace File String: **SETTRACE PRINT** (*string*\$)
3. Set Tracing of Windows Host Program: **SETTRACE SERVER**

Where

chan	Channel or logical file number of the file to which the trace output will be written.
PRINT	Keyword indicating the trace of a file string.
SERVER	<i>For internal use only.</i>
string \$	Character string to be displayed in the trace output. String expression. This is treated as a remark if no SETTRACE or Trace Window is active.

Description

When you use the **SETTRACE** directive, ProvideX lists all statements as they are executed until an **ENDTRACE** is encountered.

The **END** and **STOP** directives halt tracing. Tracing is suppressed during the execution of a password-protected program.

When tracing, ProvideX lists the statements either to the terminal or to a file (if you designate a channel in a **SETTRACE** directive). If you use a file, you have to open it first and if it's a serial file, you have to lock it.

It can be confusing if an error occurs in the output/trace file, since the error will be reported as an error for the current statement being traced, which itself may not be in error.

See Also

[ENDTRACE End Trace Output, p.118.](#)

Example

```
0010 I=3
0020 A=A*A
0030 I=I-1; IF I<>0 THEN GOTO 0020
->SETTRACE
->run
0010 I=3
0020 A=A*A
0030 I=I-1; IF I<>0 THEN GOTO 0020
0020 A=A*A
0030 I=I-1; IF I<>0 THEN GOTO 0020
0020 A=A*A
0030 I=I-1; IF I<>0 THEN GOTO 0020
->endtrace
```

SHORT_FORM Directive *Use Short Variable Names*

Format **SHORT_FORM**

Description Use the **SHORT_FORM** directive to have the compiler input-parsing accept only short variable names (emulation/compatibility mode). Its complementary directive is **LONG_FORM** which allows long variable names.

Programs can be written in either **SHORT_FORM** or **LONG_FORM** or in a combination of both and can run in either mode.



Note: This directive is included for compatibility with other languages.

See Also [LONG_FORM Use Long Variable Names, p.201](#),
['SF' System Parameter, p.686](#).

Example

```
SHORT_FORM
-:ABCDE$="ABCDE$ IS NOT OK"
Error #20: Syntax error ...BCDE$="ABC... (Long variable name not accepted)
-:A$="A$ IS OKAY"
-:LONG_FORM
-:LONG_NAME$="LONG_NAME$ IS OK"
-:
```

SHOW Directive

Show Control

Format **SHOW** *ctl_id*

Where:

ctl_id Unique CTL value for the hidden object or custom control (**BUTTON**, **LIST_BOX**, etc.)

Description Use the **SHOW** directive to re-display a given hidden control object so that the user can see it and the program can set the focus on it.

(The **HIDE** directive hides a control from display. While the control is hidden it is still active for the purposes of reading/writing but the user cannot see it, nor can the program set focus on it.)

See Also [HIDE Hide Control, p.156.](#)

Example

```
0010 BUTTON BTN.VISA,@(10,10,10,2)="&Visa Info"  
0100 READ (1,KEY=CUST_ID$)IOL=1000  
0110 IF CST.TERM$="CHQ" THEN HIDE BTN.VISA ELSE SHOW BTN.VISA
```

SORT Directive

Create File for Sorting

Format `SORT filename$,max_keysize[,max_rec][,ERR=stmtref]`

Where:

filename\$ Name of the **SORT** file to create. String expression.

max_keysize Maximum key size to be maintained for this file. Numeric expression.

max_rec Estimated number of records that the file is to contain. Default is no initial allocation of file space, with no limit on final size. Numeric expression.

stmtref Program line number or label to transfer control to.

Description Use the **SORT** directive to create a data Sort file (a file with a key but no data record portion). You must include the size of the key along with the name. The maximum Sort key size allowed is 127. If the given filename already exists, ProvideX returns Error #12: File does not exist (or already exists).

The type of file created can be determined by setting the '**KF**'= [System Parameter, p.671](#). If '**KF**' is not zero then the Sort file will be created as an EFF file. Only through use of the '**KF**' parameter will Sort files be created as EFF files

WindX supports the use of this directive via the [WDX] tag; e.g., SORT "[WDX]somefile.ext" ... for more information, see [\[WDX\] Direct Action to Client Machine, p.801](#).

See Also [File Types, User's Guide](#).

Examples 0010 SORT "CSTFLE" ,6,100 creates a file with the following structure:

```
Keyed file: C:\Program Files\Sage Software\ProvideX\CST\CSTFLE
Maximum Record size .....: 0 (Sort file)
Maximum # records .....: 100
Current # records .....: 0
Size of key block .....: 2048 bytes
External key size .....: 6
```

0010 SORT A\$+"_" +B\$,10,100,ERR=1090 creates the following structure:

```
Keyed file: C:\Program Files\Sage Software\ProvideX\CST\CST_TST
Maximum Record size .....: 0 (Sort file)
Maximum # records .....: 100
Current # records .....: 0
Size of key block .....: 3072 bytes
External key size .....: 10
```

START Directive

Restart ProvideX

Formats

1. *Restart Session*: **START** [*max_mem*[,*prog_name*]][,*ERR=stmtref*]
2. *Launch Separate Task*: **START** *prog_name*[,*term_id*][,*ERR=stmtref*]

Where:

<i>max_mem</i>	Number of 1024-byte units of memory you want to limit ProvideX to using.
<i>prog_name</i> \$	Character string defining the initial program to be loaded and run. String expression (maximum 256 characters).`
<i>term_id</i>	Terminal ID in FID(0) format for the new session. Numeric expression.
<i>stmtref</i>	Program line number or label to transfer control to.

Description

Use the **START** directive to re-initialize the current session or to launch a completely new session. **START** can be used to specify the maximum amount of memory to be allocated to the user and optionally the program to **LOAD** and **RUN**. If you omit the memory size, ProvideX uses the current memory limits.



Note: Use of the **START** directive causes a session to disconnect if ProvideX is running under WindX over a client-server connection.

Format 1: Restart Session

START [*max_mem*[,*prog_name*]][,*ERR=stmtref*]

This format completely re-initializes the current session by closing all files, clearing all user data (including Global variables), and clearing the current program. All memory currently allocated to the session is released to the system and a new limit is defined.

Examples:

```
->START 10 ! Re-initializes with 10KB
->START 20, "INVGEN" ! Runs program INVGEN with 20KB
->START ! Re-initializes with same memory size.
```

To simplify conversion processes, you can set the '**QS**' parameter to on, so that a statement like **START** *nnn*, "*this_prog*" only clears local variables (the same as a **BEGIN**) and starts the specified program. We recommend that you use **BEGIN**; **RUN** "*this_prog*" to start a new program set. Use **START** to completely restart the session.



Note: In ProvideX, the memory size might be different from other Business Basics due to the nature of memory management for variables, arrays, program sizes, etc. You can set the '**IZ**' system parameter to have ProvideX ignore any memory size restrictions.

Format 2: Launch Separate Task

START *prog_name*[,*term_id*][,*ERR=stmtref*]

You can launch a new session, separate from the task issuing the **START** command. The contents of *term_id* (terminal identifier) will be assigned to the **FID(0)** value of the new session.

STATIC Directive *Add Local Properties at Runtime*

Format **STATIC** *varlist*

Where:

varlist List of variables for use within an object.

Description In *Object Oriented Programming* (OOP), the **STATIC** directive is used to create variables for use within an object at runtime — basically, **STATIC** is used to extend the list of **LOCAL** *properties*. This allows for the dynamic creation of variables which are visible to operations within an object, yet which are not directly accessible to outside code; e.g.,

```
STATIC A$,B, IOL=IOL(1)
```

All the named variables will be **LOCAL** so that their values will remain current while executing logic within the object.

Using the **STATIC** clause, an object that is accessing a data file with an embedded IOList will be able to handle extensions to the IOList going forward. This allows the object to then read a record from the file and have the data elements remain available for subsequent calls to the object's *methods*.

Duplicate variables are ignored; i.e., if a variable has already been declared **STATIC**, is a **PROPERTY**, or has been declared **LOCAL** for an object, then no error is reported.

Static variables will include their associated arrays; therefore, `STATIC A$` also means that `A$[1]`, . . . are all static.



Note: Ensure that the **STATIC** declaration occurs before the variables are used - static variables will only take effect on references that follow their declaration.

See Also

[Object Oriented Programming, p.22](#)
[DEF CLASS Define Object Class, p.65](#)
[DROP CLASS Delete Class Definition, p.102](#)
[DROP OBJECT Delete Object, p.104](#)
[LOAD CLASS Pre-Load Class Definition, p.195](#)
[RENAME CLASS Change Name of Class, p.283](#)
[NEW\(\) Function, p.489](#)
[REF\(\) Function, p.512](#)
[Data Integration, User's Guide](#)

STOP Directive

Halt Program Execution

Format **STOP**

Description Use the **STOP** directive to halt the currently running program. If the current program is a subprogram, then control is immediately passed back to the calling program. Otherwise all open files are closed, a **RESET** operation is performed, and the next location counter is set to the start of the program.

If an application is invoked directly by an operating system command that specifies a lead program, then the **STOP** directive performs the function of a **QUIT** and automatically returns the user to the operating system. If the application is **RUN** from Command mode, ProvideX returns to Command mode.

When you use the **STOP** directive in a compound statement, it must be the final directive. (*Exception:* A remark can follow the **STOP** directive.)

The **END** directive is functionally identical to the **STOP** directive.

See Also [QUIT Terminate ProvideX, p.264](#),
[RELEASE Terminate ProvideX, p.279](#),
[END Halt Program Execution, p.113](#).

SWITCH..CASE Directive

Branch Control

Format	SWITCH <i>expression</i> CASE <i>range_1</i> [... CASE <i>range_n</i>] [BREAK] [DEFAULT] ..END SWITCH
	<i>Where:</i>
CASE <i>range_1</i> .. CASE <i>range_n</i>	List of string or numeric values for comparison with <i>expression</i> , used to define a branch point.
BREAK	Optional directive defining immediate exit from the CASE structure.
DEFAULT	Optional directive defining a default branch point should no matching CASE be found.
END SWITCH	Directive required to end branching sequence.
<i>expression</i>	String or numeric expression to test for branching to subsequent CASE statements.

Description The **SWITCH** directive defines an *expression* that will direct control to one of multiple branch points. The results of the **SWITCH** *expression* is compared with values in each **CASE** statement to determine a branch. If a match is found, execution continues with statement(s) after the matching **CASE** (until the next **BREAK** or **END SWITCH**). If there are no matches in any of the **CASE** statements, control falls through to the **DEFAULT** clause (if present), and the statements that follow are executed automatically.



Note: Since branch controls are maintained on the stack, you must not break within another stacked context. When ProvideX encounters **EXITTO** or **RETURN** within the scope of a **SWITCH**, it removes the current **FOR**/**GOSUB**/**WHILE**/**REPEAT** entry from the stack, but does not exit the **SWITCH**.

See Also [IF..THEN..ELSE Test Condition, p.157](#)
Decision Structures, User's Guide

Example

```

00100 PROCESS_TAXCODE:
00110  LiquorTax=0,SalesTax=0,ServiceTax=0
00120  SWITCH UCS(TaxCode$)
00130  CASE "X","Z" ! two codes are tax exempt
00140  BREAK ! stop processing for case "X" here
00150  CASE "L" ! liquor pays all liquor,sales and service tax
00160  LiquorTax=cost*LiquorTaxRate
00170  ! no break here, logic falls through
00180  CASE "S" ! pays sales and service tax
00190  SalesTax=cost*SalesTaxRate
00200  ! no break here, logic falls through
00210  CASE "V" ! service tax
00220  ServiceTax=cost*ServiceTaxRate
00230  BREAK ! end processing for this case and any that fell through
00240  DEFAULT ! enter here if case not found
00250  MSGBOX "Unknown tax code","Error"
00260  END SWITCH
00270  TotalTax=LiquorTax+SalesTax+ServiceTax
00280  RETURN


```

SYSTEM_HELP Directive Invoke Windows Help

Format

1. *Invoke Standard Help*: **SYSTEM_HELP** *help_path\$* [, *help_key\$*] [, **ERR=stmtref**]
2. *Invoke Application Help*: **SYSTEM_HELP** " *help_msg* ", " , *ctl_id* [, **ERR=stmtref**]

Where:

<i>ctl_id</i>	Unique logical identifier for object. Numeric expression. Use integers, range: -32000 to +32000. Avoid integers that conflict with keyboard definitions (e.g., using 4 can cancel CTL=4 for the  key).
<i>help_key\$</i>	Key to start with in the help file. This string expression must exactly match an entry unless the first character is: ? to denote a partial key # to denote that the key is the help index.
<i>help_path\$</i>	Pathname of the help file. Maximum string size 8kb.
<i>'help_msg'</i>	Application supplied message text, prefixed with an apostrophe.
<i>stmtref</i>	Program line number or label to transfer control to.



Note: This directive only functions in Windows or with the WindX terminal driver. When using WindX, the help file must be resident on, or directly accessible to, the remote workstation.

Description Use the **SYSTEM_HELP** directive to invoke standard or application-supplied Help.

Format 1: *Invoke Standard Help*

SYSTEM_HELP *help_path\$* [, *help_key\$*] [, **ERR=stmtref**]

Use this format to invoke the standard Windows Help system and CHM-based Help files. Help files must be created in accordance with the Windows standard.

Indicate the key to start with in the Help index. (Optional.) If a partial key (?) is used, the Help subsystem tries to find the entry in the Help index that most closely matches it. If the first character is a # then the key is considered the Help index for the file. If it is neither ? nor #, the key must be an exact match for an entry in the Help index.

Example:

```
1000 INPUT "Enter customer ID:", C$
1010 IF CTL=1 THEN SYSTEM_HELP "OENTRY.HLP", "Client"; GOTO 1000
```

SYSTEM_HELP allows you to use files other than those with a .HLP extension. ProvideX passes these directly to the Windows Shell command for processing, allowing Windows to apply normal file associations to automatically launch the appropriate application.

Preface the second argument with a # and append it to the first argument for URL handling; e.g.,

```
SYSTEM_HELP "http://www.pvx.com/support.htm", "mymark"
```

would be `http://www.pvx.com/support.htm#mymark`.

Passing this to **SYSTEM_HELP** will automatically launch the default browser (if any) and jump to the web page requested.

Format 2: *Invoke Application-Supplied Help*

```
SYSTEM_HELP " 'help_msg'", "", ctl_id [,ERR=stmtref]
```

If this format is used, the text following the single quote in the Help message is displayed in a popup Help box; e.g.,

```
SYSTEM_HELP "'Press Me'", "", 10
```

SYSTEM_JRNL Directive File System Journalization

- Formats
1. *Open System Journal*: **SYSTEM_JRNL OPEN** *journal*\$
 2. *Close System Journal*: **SYSTEM_JRNL CLOSE**
 3. *Enable Journalization*: **SYSTEM_JRNL ENABLE** *filename*\$
 4. *Disable Journalization*: **SYSTEM_JRNL DISABLE** *filename*\$
 5. *Add Entry to Journal File*: **SYSTEM_JRNL WRITE** *var*\$
 6. *EFF Begin Transaction*: **SYSTEM_JRNL BEGIN**
 7. *ODB/OCI/DB2 Auto-Commit Control*: **SYSTEM_JRNL AUTO** {ON | OFF}
 8. *ODB/OCI/DB2/EFF Commit Transaction*: **SYSTEM_JRNL SAVE**
 9. *ODB/OCI/DB2/EFF Roll Back Transaction*: **SYSTEM_JRNL RESTORE**
 10. *Dirty File Indicator*: **SYSTEM_JRNL DIRECTORY** *directory*\$
 11. *Track Updates*: **SYSTEM_JRNL UPDATE** *log*\$

Where:

<i>directory</i> \$	Name of common directory where tracking files are to be created.
<i>filename</i> \$	Data file identified for journalization.
<i>journal</i> \$	Name of the serial file being used as the journal. String expression. First, create the journal as an empty serial file; via SERIAL <i>journal</i> \$
<i>log</i> \$	Log file containing entry for each keyed/EFF and indexed file updated.
<i>var</i> \$	String containing data to be added to an open journal.

Description Use **SYSTEM_JRNL** to have ProvideX log all updates to specified data files.

See Also [SERIAL Create a Sequential File, p.302.](#)

Format 1: *Open System Journal*

SYSTEM_JRNL OPEN *journal*\$

This format opens the journal (serial file); e.g.,

```
SERIAL "MYJRNL"
SYSTEM_JRNL OPEN "MYJRNL"
```

Journal Contents

The physical journal file is a binary file that has a special format designed for quick write access and retrieval. Each record has a 16-byte header consisting of:

- 1 Type of record indicated by the letters:
 - O Open system journal file
 - C Close system journal file
 - o Open file (1st update request)
 - c Close file
 - B Before image of record
 - A After image of record

	D	Delete record
	E	Erase file
	P	Purge file
	U	User transaction
2-4		Record size in bytes
5-8		Time of day when record was written in seconds past 01/01/1970 GMT
9-12		Address of prior record for session
13-16		Address of related record; e.g., After/Before image points to file Open record. Open file record points to User login record

The remainder of the record is the before/after image of the data. The file itself has a four-byte header with the address of the last record on the file. Individual entries written to a journal file are linked together by their prior and related record addresses.

The first record in a session will start with an O record type (**SYSTEM_JRNL OPEN**). This record contains addition information including the user ID (**WHO/UID**) and the **FID(O)** value separated by a *semicolon*. Closing a journal file will generate a C record which links back to the original O record for the session.

Any files opened during the session will be written as an o type record which will contain a prior record address that links back to the start of the session (or the O record). Closing or updating a file (c, A, B, D type records) will link back to the file open entry (o record) to identify which file the update pertains to.

Additional data contained within an A, B or D type record is broken down as follows:

Byte 1	Length of the key/index value which follows immediately, padded with \$00\$ to the full length of the key definition (use <code>asc(data\$(1,1))</code> to determine the length)
Bytes 2,n	The actual key/index value
Bytes 2+n,2	Length of the data itself (use <code>dec(\$00\$+data\$(2+n,2))</code> to retrieve the length)
Bytes 2+n+2,x	The actual data

Format 2: *Close System Journal*

SYSTEM_JRNL CLOSE

This format closes the open system journal.

Format 3: *Enable Journalization*

SYSTEM_JRNL ENABLE filename\$

Use this format to enable journalization for the specified data file (Keyed or Indexed files only). Enabling journalization sets a bit in the file header for the specified file.



Note: If you have enabled journalization for a data file, you cannot access that file unless a journal file is open. Attempting to open a file set for journalization without an open journal file will result in Error #69: No Journalization file open.

Format 4: *Disable Journalization*

```
SYSTEM_JRNL DISABLE filename$
```

Use this format to disable journalization for the specified data file.

Format 5: *Add Entry to Journal File*

```
SYSTEM_JRNL WRITE var$
```

This format adds the contents of *var\$* to the currently open journal (serial file).

Format 6: *EFF Begin Transaction*

```
SYSTEM_JRNL BEGIN
```

This format starts a transaction for Enhanced File Format (EFF) files.

Format 7: *ODB/OCI/DB2 Auto-Commit Control*

```
SYSTEM_JRNL AUTO {ON | OFF}
```

This format is used to control auto-commit mode of an ODBC, Oracle, or DB2 database connection. Auto-commit means that all updates to the database are made immediately and cannot be rolled back in case of an error. See previous note.

SYSTEM_JRNL AUTO OFF will disable the auto-commit mode of operation. All updates to the database are deferred until the application successfully ends or a **SYSTEM_JRNL SAVE** directive is issued. Also, **SYSTEM_JRNL AUTO ON** re-enables Auto-commit mode.

Format 8: *ODB/OCI/DB2/EFF Commit Transaction*

```
SYSTEM_JRNL SAVE
```

This format will commit an ODBC, Oracle, or DB2 database transaction if auto-commit mode is disabled; **ERROR #15** is reported if a database error occurs.

Pending EFF transactions will be committed as well.

Format 9: *ODB/OCI/DB2/EFF Roll Back Transaction*

```
SYSTEM_JRNL RESTORE
```

This format will roll back an ODBC, Oracle, or DB2 database transaction if auto-commit mode is disabled; **ERROR #15** is reported if a database error occurs.

Pending EFF transactions will be rolled back as well.

Format 10: *Dirty File Indicator*

SYSTEM_JRNL DIRECTORY *directory*\$

This format is used to track potential file corruption (*dirty files*) by maintaining a list of all the files that have been opened and modified during a ProvideX session. ProvideX creates a single tracking file per active process in *directory*\$.

When a session terminates normally, it deletes its tracking file; therefore, the existence of a tracking file means there is either a ProvideX task currently active, or that a ProvideX task has terminated abnormally (due to software fault or operating system failure).

Specifying a null directory name will *disable this feature*. If the directory name specified does not already exist, an Error #12: File does not exist (or already exists) will be generated. An Error #13: File access mode invalid is reported if a file of the same name exists, but not as a directory.

Tracking Files

Each tracking file (log) is created with a unique name using the following format:

username.MMDDHHMMSS.log

Where:

username Name for the session. In the case of client-server applications, the workstation name (if present) or IP address will be used, otherwise the user name for the host process will be used. No attempt is made to determine the true workstation end-user name – only the workstation name, as returned from the socket's 'GetPeerName' function, will be used.

MMDDHHMMSS Date and time that the log started.

The tracking file contains a list of all the data files that have been physically updated and not yet closed by the ProvideX process. Every time a file has an update issued against it, the system will add the pathname to the tracking file. Each line will be separated by a standard (OS dependant) end-of-line sequence.

Multiple occurrences of the same file may exist in the tracking file as entries are maintained based on open channels.

The tracking file is updated to indicate when a file is being updated prior to the execution of the **WRITE** or **REMOVE** directives. When a file that was logged is closed, its entry from the log file will be removed.

To help ensure that the contents of the tracking file are accurate, ProvideX will close the tracking file after each write.\



Note: Only Keyed files (EFF, VLR and FLR) and Indexed files are tracked.

Format 11: *Track Updates***SYSTEM_JRNL UPDATE** *log\$*

SYSTEM_JRNL UPDATE writes entries to the specified *log\$* file for every ProvideX keyed/EFF and indexed file updated during a session. Each entry consists of the name of the file being updated for each channel a file is updated on. A subsequent **SYSTEM_JRNL UPDATE** resets the status of whether a file has been included in a previous log allowing all updates to be tracked from a specific point in the application.

Examples

The following code sample illustrates parsing of a **SYSTEM_JRNL** file:

```

00010 ! DumpJrnl - Sample routine to parse a System_Jrnl file
00020 ! Note: Time is in GMT
00030 OPEN (1,ISZ=1)"jrnl.dat"
00040 READ RECORD (1,IND=0,SIZ=4,ERR=0500)filesize$
00050 LET filesize=DEC($00$+filesize$)
00060 PRINT "Size of Journal file is: ",filesize
00070 LET nextindex=4
00100 !^100
00110 IF nextindex>filesize \
      THEN GOTO 0500
00120 READ RECORD (1,IND=nextindex,SIZ=16,ERR=0500)header$
00130 LET type$=header$(1,1)
00140 LET recsize=DEC($00$+header$(2,3))
00150 LET time=DEC($00$+header$(5,4))
00160 LET priorrec=DEC($00$+header$(9,4))
00170 LET relatedrec=DEC($00$+header$(13,4))
00180 LET datarec$="";
      IF recsize>16 \
      THEN READ RECORD
      (1,IND=nextindex+16,SIZ=recsize-16,ERR=0500)datarec$
00190 LET nextindex+=recsize
00300 !^100 - Format & display
00310 LET disprec$=SUB(SUB(datarec$,SEP,"~"),ESC,"~")
00320 LET days=INT(time/86400);
      IF (days*86400)>time \
      THEN days--
00330 LET time=time-days*86400
00340 LET sv_by=PRM('BY');
      SET_PARAM 'BY'=1970
00350 LET t$=DTE(days,time/3600:"%Ms%Dz/%Ys-%hz:%mz:%sz%p")
00360 SET_PARAM 'BY'=sv_by
00370 !
00380 PRINT t$," Type: ",type$," Prior Rec:",priorrec:"####,##0","
      RelatedRec:",relatedrec:"####,##0"," RecSize",recsize:"###,##0","
      Record:",disprec$
00390 GOTO 0100
00400 !
00500 !^100

```

The following illustrates use of the **SYSTEM_JRNL** for tracking potential file corruption (dirty file indicator):

```
Dir$="/SysJrnl.Dir/"; DIRECTORY Dir$,ERR=*NEXT
  WorkFile$="WorkFile.Dat"; KEYED WorkFile$,10,0,-256,ERR=*NEXT
!
! Generate a log file
SYSTEM_JRNL DIRECTORY Dir$
OPEN (UNT)WorkFile$; WorkFile=LFO
WRITE (WorkFile,KEY="Test")"Test","Record"
!
List$=""
SELECT Log$ FROM Dir$ WHERE POS(".log"=Log$)
  SELECT File$ FROM Dir$+Log$
    IF POS(File$+SEP=List$) \
      THEN CONTINUE \
      ELSE List$+=File$+SEP
    PRINT "Checking: ",File$," ",
    CALL "*ufac",ERR=*NEXT,File$,1; PRINT "Okay"; CONTINUE
    PRINT File$,":",MSG(ERR)
  NEXT RECORD
NEXT RECORD
!
CLOSE (WorkFile)
```

TABLE Directive*Define Translation Table*

Format **TABLE** *hex-table*

Where:

hex-table Hex conversion table which consists of multiple hex digit pairs.

Description Use the **TABLE** directive to define the translation table ProvideX is to use to convert data from one format to another. The translation table to use is identified via the **TBL=** option in I/O operations (**READ**, **WRITE**, etc.) or within the **TBL()** function. Use the first pair of hex digits to define the conversion mask, the remaining hex pairs to define the converted data bytes.

A translation table is defined by a statement with the **TABLE** directive followed by a series of hex digits. Each pair of hex digits represent a single byte. The first byte in the table (first two hex digits) is the translation and mask. It is **ANDed** with the hex value of each byte as it is being converted. The results of this **AND** operation is then used as an offset (base zero) into the rest of the transliteration table. The byte at this offset is the resultant data byte which is either returned in the case of the **TBL()** function and file input, or is written to a file in the case of a file output.

Translation tables are usually used to convert one character set to another such as ASCII to EBCDIC or for data encryption.

See Also [TBL\(\) Function, p.532](#)

Example 0010 TABLE 0F303132333435363738394142434444546

The above table would result in the following:

Input		After	Output	
<i>ASCII</i>	<i>Hex</i>	<i>AND</i>	<i>ASCII</i>	<i>Hex</i>
A	41	01	1	31
B	42	02	2	32
z	7A	0A	A	41
[5B	0B	B	42
P	50	00	0	30

TRANSLATE Directive *Translate Contents of Variable*

Formats

1. *Translate From String*: **TRANSLATE** *var\$,table\$,offset*
2. *Translate From Variable to Variable*: **TRANSLATE** *to_var\$,from_var\$,offset*
3. *Translate Single Character*: **TRANSLATE** *var\$,to_char\$,from_char\$*
4. *Translate Character via Hex Value*: **TRANSLATE** *var\$,hex_string\$*

Where:

<i>from_char\$</i>	Character to be searched_for and replaced.
<i>from_var\$</i>	Conversion table to be used as the new value(s) in the translation. String expression.
<i>hex_string\$</i>	Formatted hex string indicating characters and replacement characters.
<i>offset</i>	Base value to subtract from the characters in the variable to calculate the offset into the table during conversion. Numeric expression.
<i>table\$</i>	Conversion table to be used as the new value(s) in the translation. String expression.
<i>to_char\$</i>	New character to replace the original <i>from_char\$</i> . All instances of the <i>from_char\$</i> in the <i>var\$</i> string are replaced with the <i>to_char\$</i> .
<i>to_var\$</i>	Variable in which the translation is performed. Starting at the offset, <i>to_var\$</i> receives the translation of its own old values to the corresponding new values (from the table or <i>from_var\$</i> string).
<i>var\$</i>	String variable to be translated.

Description

Use the **TRANSLATE** directive to convert/translate a string variable on a character-by-character basis using a conversion table or value. Use this directive to convert data from one character set to another (e.g., ASCII to EBCDIC).

Format 1: *Translate From String*

TRANSLATE *var\$,table\$,offset*

Use this format to replace a character in a string variable with a character from a table string/expression. ProvideX performs the conversion by taking each character from the variable, subtracting the offset from its binary value, and using the result as an offset into the table whose character will then replace the original character in the variable. When ProvideX computes the offset into the table, it will treat an offset of zero as the first character from the table. If the result of the subtraction is an offset that exceeds the length of the table, then no conversion takes place for the character.

Example:

```

0110 LET A$="ABCD"
0130 TRANSLATE A$, "ZYXWVUTS" ,DEC("A")
0140 PRINT A$
->GOTO 110
->RUN
ZYXW

```

Format 2: Translate From Variable to Variable**TRANSLATE to_var\$,from_var\$,offset**

Use this format to copy the character-by-character values from one variable to another variable up to the length of the shorter variable. In this format, ProvideX also uses a numeric starting offset into the target table.

Example:

```

0110 LET A$="1234567890"
0120 LET B$="ABCD"
0130 TRANSLATE A$,B$,DEC("3")
0140 PRINT A$
->GOTO 110
->RUN
12ABCD7890

```

Format 3: Translate Single Character**TRANSLATE var\$,to_char\$,from_char\$**

Use this format to replace all instances of a given character in a string variable with another character.

Example:

```

0110 LET STRVAR$="ABxDEFG xxx"
0120 TRANSLATE STRVAR$, "C" , "x"
0130 PRINT STRVAR$
-:run
ABCDEFG CCC

```

Format 4: Translate Character via Hex Value**TRANSLATE var\$,hex_string\$**

Use this format to replace a single character in a string variable with designated character(s). *hex_string\$* consists of a string of hexadecimal values indicating the character to be replaced, the number of characters to replace that character, then the replacement character(s).

Examples:

```
TRANSLATE R$, $0A020D0A$
```

The line-feed character (`$0A$`) will be replaced by two (`02`) characters: carriage-return and line-feed (`$0D0A$`).

Multiple sets of characters may be translated by extending the translation string; e.g.,

```
TRANSLATE R$, $0A020D0A090120$
```

In this case, the line-feed character (`$0A$`) will be replaced by the two (`02`) carriage-return and a line-feed characters (`$0D0A$`), and the tab character (`09`) will be replaced by one (`01`) space character (`20`).

TRISTATE_BOX Directive

Control Tristate Box

Formats

1. *Define/Create*: `TRISTATE_BOX [*]ctl_id,@(col,ln,wth,ht)=contents$[,ctrlopt]`
2. *Remove*: `TRISTATE_BOX REMOVE [*]ctl_id[,ERR=stmtref]`
3. *Disable/Enable*: `TRISTATE_BOX {DISABLE | ENABLE} [*]ctl_id[,ERR=stmtref]`
4. *Force Focus*: `TRISTATE_BOX GOTO [*]ctl_id[,ERR=stmtref]`
5. *Logical Push/Release*: `TRISTATE_BOX {ON | OFF} [*]ctl_id[,ERR=stmtref]`
6. *Read Activation State*: `TRISTATE_BOX READ [*]ctl_id,state$[,ERR=stmtref]`
7. *Update*: `TRISTATE_BOX WRITE [*]ctl_id,state$[,ERR=stmtref]`

Where

- * Optional. Use a leading asterisk to denote a *global* tristate box.
- @(col,ln,wth,ht) Position and size of tristate box region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines are for the total area (box plus text/description). Use line coordinate -1 to display the tristate box on the tool bar.
- contents\$ Text/pictures for the tristate box. Both {bitmap} and {icon} images are supported. String expression. See [TRISTATE_BOX contents\\$, p.346](#).
- ctl_id Unique logical identifier for a tristate box control (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the **F4** key) or [Negative CTL Definitions, p.817](#). Use this value with the apostrophe operator to access various [Tristate Box Properties](#).
- ctrlopt Control options. Supported options for **TRISTATE_BOX** include:
 - ERR=stmtref** Error transfer
 - FNT="font,size[,attr]"** Font name, size, optional properties
Refer to the ['FONT' Mnemonic, p.609](#) for details.
 - MSG=text\$** Message string.
 - MNU=ctl** CTL value associated with right-click menu event.
 - OPT=char\$** (See [TRISTATE_BOX OPT= Settings; p.345](#))
 - OWN=name\$** Name assigned for automated testing of this control.
 - TBL=char\$** Single character translation.
 - TIP=text\$** Mouse pointer message.
To change the colour, refer to the ['TC'= System Parameter, p.689](#).
- state\$ Current state of the **TRISTATE_BOX ON** or **OFF** or *other* (e.g., as is).
- stmtref Program line number or label to transfer control to.

TRISTATE_BOX OPT= Settings:

Available attribute/behaviour settings are listed below. Some characters may be combined. Invalid settings are ignored.

- "<" *Bitmap Left.* Places bitmap left of text.
- ">" *Bitmap Right.* Places bitmap right of text.
- "^" *Drop-down.* Adds drop-down functionality.
- "*" *Default.* Defines tristate box as default.
- "A" *Auto.* Generates a CTL value signal for every character the user enters.
- "B" *Bitmap.* Has a bitmap whose width is divided into four images. Use this attribute to custom design tristate boxes of any colour, style or shape by controlling the bitmap image that appears. Each of the four divisions represents what a button will look like in a particular state:
 - 1st quarter: Bitmap image when button is disabled.
 - 2nd quarter: Bitmap image when button is in normal (released) state.
 - 3rd quarter: Bitmap image when the mouse is over the button.
 - 4th quarter: Bitmap image when the button is pressed.
- "D" *Disabled.* Tristate box is grayed out and is not accessible to the user.
- "F" *Flat.* Tristate box shows no raised outline unless the mouse is over the button or the button is pushed.
- "f" *Flat-No Shift.* Same as "F", but will not shift when pressed.
- "G" *Global.* Keep active when focus changes to new/non-concurrent window. When using secondary commands (**REMOVE** or **SET_FOCUS**) on controls created with **OPT="G"** identify the control by prefixing the CTL value with an asterisk; e.g.,
`TRISTATE_BOX 100,@(10,10,10,1)="Global",OPT="G"`
`TRISTATE_BOX REMOVE *100`
- "H" *Hide.* The user can't see the tristate box even though it is still active and accessible through the program.
- "S" *Signal Only.* ProvideX generates a CTL value, but does not shift focus to the tristate box automatically (the default), but only when focus is explicitly passed to it. Use this to have a tristate box act like a function key.
- "s" *Scroll.* Tristate box can scroll within a resizable/scrollable dialog box.
- "T" *Transparent.* Tristate box is "see-through" to window data behind.
- "U" *Underscore.* Text is underlined.
- "V" *Hovertext.* Indicates that text will change colour when mouse is over control.
- "Y" *System Tray.* Places an icon in the *Taskbar Notification Area*.

Combined options can be used to create several different tristate box types. The "f", "T", and "U" options provide the ability to turn tristate boxes into *hotspots*. This allows for clickable areas on bitmaps or hyperlinked text in dialogues; e.g.,

- "VTf" Creates a general hotspot.
- "VUTf" Creates an HTML-like hotspot (e.g., URL hyperlink).
- "F^" Creates a word-style toolbar with drop list

Description

Use the **TRISTATE_BOX** directive to create/control a tristate box on the screen or to place an icon in the **Taskbar Notification Icon** (see *User's Guide*, p.155). A tristate box is a check box in which the user can toggle between three states: **ON** to select an option, **OFF** to disable it, or your choice of a *third* state. Refer to the Tristate Box example under **Format 1: Define/Create**, p.347, for an illustration. See **Chapter 7. Control Object Properties**, p.701 for a list of properties you can use with tristate boxes.

TRISTATE_BOX contents\$

The *contents\$* string expression defines the text or picture to appear on the tristate box. In the text, you can use an ampersand "&" preceding a character to identify it as a hot key the user can press in conjunction with the **Alt** key to activate the tristate box from the keyboard.

Using Images

When adding an image to a tristate box, enclose the image name in curly braces. Use a leading exclamation point (!) to identify the image as internal, or specify the relative path and filename to access an image file that is external. There are no icons in the ProvideX executable and ProvideX does not support retrieving icons from either resource libraries or other system DLLs /executables. For more information on the options available for displaying internal/external images and the recognized image file types, see **Images and Icons**, p.153 in the *User's Guide*.

When you use text as well as images, the relative positions of the image and the text set their relative placement. The following are example *contents\$* expressions:

```
"{!Add}Add" ! Displays the {!Add} bitmap in front of the text "Add"  
"Delete{!Del}" ! Displays the {!Del} bitmap after the text "Delete"
```

If your string expression includes three bitmaps separated by a vertical bar inside a single set of curly braces, the first will be displayed when the tristate box is in its normal state, **OFF**, the second while the tristate box is **ON**, and the third when the tristate box is in a third *other* state.

You can also use the `OPT="B"` clause for a *Bitmap Button* to display different images for different states.

Tristate Box Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a tristate box are described in **Chapter 7. Control Object Properties**, p.707.

Format 1: *Define/Create*

```
TRISTATE_BOX [*]ctl_id,@(col,ln,wth,ht)=contents$[,ctrlopt]
```

Use the value in *ctl_id* to give your tristate box a unique identifier. This value generates a CTL value whenever the tristate box is toggled. If *ctl_id* has a leading *asterisk* *, the tristate box is considered global (not tied to a specific window).

Examples:

```
0120 TRISTATE_BOX 102,@(2,19,12,3)=" {!File_Save}&Save"
0130 TRISTATE_BOX 103,@(22,15,12,2)="&Destroy{!Trash|!Trash_open|!Bomb_blast}"
0140 TRISTATE_BOX 104,@(22,19,12,3)=" {!Lite_Green|!Lite_Yellow|!Lite_Red}"
0150 INPUT (0,HLP="Tristate_Box")@(40,18),"Select...: ", 'CL', X$
0160 IF CTL>100 AND CTL<105 THEN TRISTATE_BOX READ CTL,B$; PRINT @(40,19),"Sele
0160:ction:",CTL,":",B$,'CL',; GOTO 0150
0170 IF CTL=0 OR CTL>=3 THEN STOP ELSE GOTO 0150
```

Format 2: *Remove*

```
TRISTATE_BOX REMOVE [*]ctl_id,@(col,ln,wth,ht)=contents$[,ctrlopt]
```

Use the **TRISTATE_BOX REMOVE** format to delete a tristate box. By default, if tristate boxes are not global, they are deleted when a window is removed/dropped or the application issues a **BEGIN**. Global tristate boxes can be removed manually or cleared by a **START**.

Format 3: *Disable/Enable*

```
TRISTATE_BOX {DISABLE | ENABLE} [*]ctl_id[,ERR=stmtref]
```

Use the **TRISTATE_BOX DISABLE** format to gray-out a tristate box so that it will be visible but *inaccessible* to users. To reactivate it, use **TRISTATE_BOX ENABLE**.

Format 4: *Force Focus*

```
TRISTATE_BOX GOTO [*]ctl_id[,ERR=stmtref]
```

Use the **TRISTATE_BOX GOTO** format to reactivate and force focus to a tristate box, ready for the next user action.

Format 5: *Logical Push/Release*

```
TRISTATE_BOX {ON | OFF} [*]ctl_id[,ERR=stmtref]
```

Use the **TRISTATE_BOX ON** format to make it appear that a tristate box selection has been made. Use the **OFF** format to make it appear that it has been released.

Format 6: *Read Activation State*

```
TRISTATE_BOX READ [*]ctl_id,state$[,ERR=stmtref]
```

The **TRISTATE_BOX READ** format returns the current state of the tristate box ("0" for **OFF**, "1" for **ON**, or "2" for *other*).

Format 7: *Update*

TRISTATE_BOX WRITE [*]ctl_id,state\$[,ERR=stmtref]

Use the **TRISTATE_BOX** format above to write/update new values for the tristate box.

See Also

RADIO_BUTTON Control Radio Button, p.265

BUTTON Control Button, p.34

CHECK_BOX Control Check Box, p.47

Chapter 7. Control Object Properties, p.701.

UNLOCK Directive *Remove Exclusive Use from File*

Format **UNLOCK** (*chan* [, **ERR**=*stmtref*])

Where:

chan Channel or logical file number of the file to be unlocked.

stmtref Program line number or label to transfer control to.

Description Use the **UNLOCK** directive to release a previously locked file. If the given file is not already locked, ProvideX returns Error #14: Invalid I/O request for file state.

See Also **LOCK Reserve File for Exclusive Use, p.200.**

Example

```
0010 OPEN (30,ERR=0100)"GLFILE"  
0020 LOCK (30,ERR=0120)  
0030 READ (30,KEY="HEProvideXR")A  
0040 IF A>0 THEN GOTO 0100  
0050 UNLOCK (30)  
0060 PRINT "Nothing on file to process.."  
0070 STOP  
0100 REM 100 Error handling
```

UNTIL Directive

End REPEAT Loop

Format **UNTIL** *expression*

Where:

expression Condition to end **REPEAT** looping when true.



Note: Refer to **REPEAT..UNTIL Repetitive Execution, p.287**, for complete syntax.

Description Use the **UNTIL** directive to define the end of a **REPEAT** loop in a program.

See Also **REPEAT..UNTIL Repetitive Execution, p.287**.

Example

```

0010 PRINT 'CS',"Standard TAX calculation..."
0020 INPUT "How much was your INCOME? $",CASH
0110 REPEAT
0120 INPUT "How much TAX did you pay? $",TAXES
0130 LET CASH=CASH-TAXES
0140 PRINT "You have $",CASH," left"
0150 UNTIL CASH<=0
0160 PRINT "Okay you have paid enough."
-:BEGIN
-:RUN
Standard TAX calculation...
How much was your INCOME? $1.98
How much TAX did you pay? $1.65
You have $ 0.33 left
How much TAX did you pay? $.33
You have $ 0 left
Okay you have paid enough.

```

UPDATE Directive *Update Existing Record in File*

Formats **UPDATE** (*chan*[,*fileopt*])*varlist*

Where:

chan Channel or logical file number of the file to which to write.

fileopt Supported file options (see also, [File Options, p.810](#)):
BSY=*stmtref* Traps Error #0: Record/file busy
DOM=*stmtref* Missing record transfer
END=*stmtref* END-OF-FILE transfer
ERR=*stmtref* Error transfer
IND=*num* Record index
KEY=*string*\$ Record key
REC=*name*\$ Record prefix (**REC=VIS**(*string*\$) can also be used)
RTY=*num* Number of retries (one second intervals)
TIM=*num* Maximum time-out value in integer seconds (support write operations for TCP channels).

stmtref Program line number or label to transfer control to.

varlist Comma-separated list of variables, literals, and **IOL=** options.

Description The **UPDATE** directive is used to update an existing record to a file (logical file number / channel). The syntax for this directive is identical to the [WRITE Directive, p.383](#); however, **UPDATE** only updates a record if it *already exists* and will return an error if the record does not exist.

UPDATE may be used against Keyed, Memory, ODBC, and OCI files. When **IND=** is used with ***MEMORY***, this directive overwrites an existing index.

See Also [WRITE Add/Update Data in File, p.383](#)
[WRITE RECORD Write Record, p.386](#)
[INSERT Insert New Record in File, p.162](#)

USER_LEX Directive Define Alternate Keywords

Format

1. *Define Alternate Keyword*: **USER_LEX** *alt_string\$=directive\$*
2. *Load Alternate Lexicon*: **USER_LEX LOAD** *filename\$*
3. *Clear Alternate Lexicon*: **USER_LEX CLEAR**

Where:

alt_string\$ Alternate keyword/symbol to be used in place of a ProvideX directive.

directive\$ Standard ProvideX directive.

filename\$ Name of a file containing an alternate lexicon. String expression.

Description

Use the **USER_LEX** directive to define alternate keywords in order to simplify conversions from other languages to ProvideX or to provide support for languages other than English.

Format 1: *Define Alternate Keyword*

USER_LEX *alt_string\$=directive\$*

Use the **USER_LEX** directive to extend the internal ProvideX compiler definitions to include a new directive or symbol.

After ProvideX executes the **USER_LEX** directive, any statements sent to the ProvideX compiler where the alternate string expression is found will be translated into the internal object code for the standard string in *directive\$*. It is mandatory that you use valid ProvideX syntax in the *directive\$*. Use spaces in the string expression to indicate that spaces can exist in the string when compiling.

Example:

To allow the use of **SPC()** in lieu of **PAD()**:

```
USER_LEX "SPC ("= "PAD ("
```



Note: The internal syntax values in the *directive\$* must include the proper spacing and all related control characters. In the above example "SPC"="PAD" alone is not acceptable. You must include the space and open parenthesis after **SPC** and after **PAD** to indicate that a space can occur between the words (**SPC** / **PAD**) and the open parenthesis.

Format 2: *Load Alternate Lexicon*

USER_LEX LOAD *filename\$*

Use the **USER_LEX LOAD** directive to load an alternate internal lexicon table. This lexicon table can be used to simplify a conversion from other programming languages to ProvideX or to provide support for languages other than English.

The lexicon table consists of all the ProvideX keywords for *directives*, *functions* and *system variables*. ProvideX supplies a utility program, *LEXEDIT, to create and maintain these tables.



Warning: Be very careful when modifying syntax tables. While it is possible to create syntax tables for other languages and to add additional syntax to the external table, the wrong changes can make your programs un-listable. (The syntax tables affect both the listing and editing of programs.) Do not attempt to create or edit the syntax tables unless you are an experienced programmer and are familiar with these tables. *Please contact your local distributor of ProvideX for assistance.*

Format 3: *Clear Alternate Lexicon*

USER_LEX CLEAR

The **USER_LEX CLEAR** directive removes an alternate internal lexicon table and restores the standard directive keywords.



VARDROP_BOX Directive

Control Variable Drop Box

Formats

1. *Define/Create*: **VARDROP_BOX** *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]
2. *Remove*: **VARDROP_BOX REMOVE** *ctl_id*[,ERR=stmtref]
3. *Disable/Enable*: **VARDROP_BOX** {DISABLE | ENABLE}*ctl_id*[,ERR=stmtref]
4. *Hide/Show*: **VARDROP_BOX** {HIDE | SHOW} *ctl_id*[,ERR=stmtref]
5. *Force Focus*: **VARDROP_BOX GOTO** *ctl_id*[,ERR=stmtref]
6. *Signal on Focus*: **VARDROP_BOX SET_FOCUS** *ctl_id*, *ctl_val*[,ERR=stmtref]
7. *Load Via Delimited String*: **VARDROP_BOX LOAD** *ctl_id*,dlm_list\$,[,ERR=stmtref]
8. *Load Via Array*: **VARDROP_BOX LOAD** *ctl_id*,array_name\${ALL}[,ERR=stmtref]
Note: The curly braces enclosing {ALL} are part of the syntax.
9. *Load/Delete Element*: **VARDROP_BOX LOAD** *ctl_id*,index,{element\$ | *},[,ERR=stmtref]
10. *Retrieve Element*: **VARDROP_BOX FIND** *ctl_id*,index,var\$,[,ERR=stmtref]
11. *Read Current String*: **VARDROP_BOX READ** *ctl_id*,var\$[,mode\$][,ERR=stmtref]
12. *Read Current Index*: **VARDROP_BOX READ** *ctl_id*,var[,mode\$][,ERR=stmtref]
13. *Write Current Selection*: **VARDROP_BOX WRITE** *ctl_id*,element\$[,ERR=stmtref]
14. *Update Using Index*: **VARDROP_BOX WRITE** *ctl_id*,index[,ERR=stmtref]
15. *Clear Current Selection*: **VARDROP_BOX WRITE** *ctl_id*, "",[,ERR=stmtref]
16. *Report All Changes*: **VARDROP_BOX AUTO** *ctl_id*[,ERR=stmtref]

Where:

@(col,ln,wth,ht)	Position and size of the variable drop box region when expanded. Numeric expressions. Column and line coordinates for top left corner width in number of columns, and height in number of lines. (Note that drop box height, when not expanded, is governed by the system and is roughly 1.5 times the standard graphic font height.)
array_name\$	Name of array to load into variable drop box. String variable followed by {ALL}.
ctl_id	Unique logical identifier for a variable drop box (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the F4 key) or Negative CTL Definitions, p.817 . Use this value with the apostrophe operator to access Variable Drop Box Properties .
ctl_val	CTL value to generate when the variable drop box gains focus.

<i>ctrlopt</i>	<p>Control options. Supported options for VARDROP_BOX include:</p> <p>ERR=stmtref Error transfer</p> <p>FNT="font,size[,attr]" Font name, size, optional properties Refer to the 'FONT Mnemonic, p.609 for details.</p> <p>KEY=char\$ Hot key</p> <p>LEN=num Maximum input length</p> <p>MSG=text\$ Message line</p> <p>MNU=ctl CTL value associated with right-click menu event.</p> <p>TBL=char\$ Single character translation</p> <p>TIP=text\$ Mouse pointer message.</p> <p>To change the colour, refer to the 'TC= System Parameter, p.689.</p> <p>OWN=name\$ Name assigned for automated testing of this control.</p> <p>OPT=char\$ Attribute/behaviour settings:</p> <p>"D" - <i>Disabled</i>. User cannot access the drop box.</p> <p>"G" - <i>Global</i>. Keep active on focus change to new/non-concurrent window.</p> <p>"H" - <i>Hide</i>. Do not display the drop box.</p> <p>"S" - <i>Signal</i>. Generate CTL value but without shifting focus.</p> <p>"A" - <i>Auto</i>. Generate CTL signal for every character entered.</p> <p>"T" - Strip trailing spaces.</p> <p>"X" - Signal on Exit.</p> <p>"s" - <i>Scroll</i>. Allow scroll within resizable/scrollable dialogue box.</p> <p>Some characters may be combined. Invalid settings are ignored.</p>
<i>dIm_list\$</i>	Delimited list of elements to load. String expressions.
<i>element\$</i>	Single element to load. String expression. Use the <i>asterisk</i> * instead, to <i>delete</i> an element. For instance, VARDROP_BOX LOAD 86,4,* will "eighty-six" (remove) element 4 from VARDROP_BOX 86.
<i>index</i>	Position of the element in the variable drop box. Numeric expression. Integers: the index of the 1 st element is 1.
<i>mode\$</i>	String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to activate the drop box (\$01\$ for MOUSE-CLICK or \$0D\$ for Enter).
<i>stmtref</i>	Program line number or label to transfer control to.
<i>var[\$]</i>	Variable to receive value. String variable for element/numeric for index.

Description

Use the **VARDROP_BOX** directive to create and control variable drop boxes on the screen. A variable drop box normally displays a single line on the screen with a **DOWN-ARROW** on the right side and allows variable input. That is, the user can select any element from a list of items associated with the variable drop box or can enter *any other value*. To view the list the user clicks on the **DOWN-ARROW**.

Because a variable drop box list is in *drop-down form*, it takes a smaller amount of space on the screen than a comparable variable list box. In addition, ProvideX automatically supplies vertical scrollbars if the number of elements overflows the drop-down box size. Combine these features to optimize screen design when display space is at a premium.

Variable Drop Box Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a variable drop box are described in [Chapter 7. Control Object Properties, p.708](#).

Format 1: Define/Create

```
VARDROP_BOX ctl_id,@(col,ln,wth,ht)[,ctrlopt]
```

Use the format above to create a variable drop box, giving it a unique *ctl_id*. When a user selects an item from a variable drop box, or enters an item that is not on the list, the associated *ctl_id* you give to the variable drop box is used to generate a CTL value. Use the **FNT=** option to establish the font for variable drop box. If you omit the font option, ProvideX uses the system default font. Use **FNT="*"** to set the font as standard text mode fixed font.

The following example creates a variable drop box that generates a CTL=100 when any item is selected from it. It's loaded with the items Cat, Dog, and Pig.

```
0010 VARDROP_BOX 100,@(2,14,12,6)
0020 VARDROP_BOX LOAD 100,"Cat/Dog/Pig/"
```

The user can select any of the three items supplied or enter any other value.

Format 2: Remove

```
VARDROP_BOX REMOVE ctl_id[,ERR=stmtref]
```

Use the **VARDROP_BOX REMOVE** format to delete a drop box.

Format 3: Disable/Enable

```
VARDROP_BOX {DISABLE | ENABLE}ctl_id[,ERR=stmtref]
```

Use the **VARDROP_BOX DISABLE** format to gray out a variable drop box so that it will be visible but *inaccessible* to users. To reactivate it, use **VARDROP_BOX ENABLE**.

Format 4: Hide/Show

```
VARDROP_BOX {HIDE | SHOW}ctl_id[,ERR=stmtref]
```

With the **VARDROP_BOX HIDE** format, the drop box remains active, but is not displayed. It is still accessible programmatically. Use the **SHOW** format to restore the display and user access.

Format 5: Force Focus**VARDROP_BOX GOTO** *ctl_id*[,ERR=*stmtref*]

Use **VARDROP_BOX GOTO** to reactivate and force focus to a variable drop box, ready for the next user action.

Format 6: Signal on Focus**VARDROP_BOX SET_FOCUS** *ctl_id, ctl_val*[,ERR=*stmtref*]

Use the **VARDROP_BOX SET_FOCUS** format to define an alternate CTL value to generate whenever focus shifts to the variable drop box.

Formats 7, 8 and 9: Load a Variable Drop Box

Use the **VARDROP_BOX LOAD** formats below to load items into a variable drop box. The element(s) can be loaded as a *delimited string*, as an *array of string elements*, or *individually*.

VARDROP_BOX LOAD *ctl_id,dlm_list*\$_[,ERR=*stmtref*]

Load List. When you load items from a *delimited string*, the last character in the string must be the delimiter; e.g.,

```
0100 VARDROP_BOX LOAD 10000, "Fox/Cat/Dog/Cow/Sheep/Horse/Pig/Elephant/Ant/"
0500 VARDROP_BOX LOAD 15000, "Fox"+SEP+"Cat "+SEP+"Dog"+SEP
```

VARDROP_BOX LOAD *ctl_id,array_name*\$_{ALL}[,ERR=*stmtref*]

Load Array. Use this format to load a complete *array* into the variable drop box. Note that the curly braces enclosing **{ALL}** are part of the syntax.

VARDROP_BOX LOAD *ctl_id,index*,{*element*\$ | *}[,ERR=*stmtref*]

Load Element. When you load a variable drop box one element at a time, the index value refers to the element before which the new element is to be inserted. For instance, if *index* is 1, the new element will be inserted before 1, at the start of the list. If *index* is 0 *zero*, the new element will be appended to the end of the list.

If you have more items in the list than will fit the physical screen size of the variable drop box, ProvideX automatically supplies scrollbars. To delete or remove a specified element from a variable drop box, use an *asterisk* * in place of the element string; e.g.,

```
VARDROP_BOX LOAD 86,4,* ! Deletes item whose index=4 from drop box 86.
```

Format 10: *Retrieve Element*

VARDROP_BOX FIND *ctl_id,index,var\$[,ERR=stmtref]*

Use **VARDROP_BOX FIND** to retrieve a specific element from a drop box.

Formats 11 and 12: *Read Current Selection*

Use the **VARDROP_BOX READ** formats to read which element the user has selected from the variable drop box. Note that you must **READ** the user's selection before your application can use it.

Use a string variable to return information on how the element was selected. The value returned will be:

\$01\$ for **MOUSE-CLICK**.

\$0D\$ for **Enter**.

After this value is read, it resets to \$00\$ (null).

VARDROP_BOX READ *ctl_id,var\$[,mode\$][,ERR=stmtref]*

Read Current Element. Use a string variable in a **VARDROP_BOX READ** to receive the value of the currently selected element. Use an optional second variable to receive the selection method (i.e., *mode\$*).

VARDROP_BOX READ *ctl_id,var[,mode\$][,ERR=stmtref]*

Read Current Index. Use a numeric variable in a **VARDROP_BOX READ** to receive the index of the element the user has selected. Once a user has made a selection, it is your responsibility to read it to return the value to your program.

Formats 13 and 14: *Write Current Selection*

Use the **VARDROP_BOX WRITE** formats described below to update the current selection in the variable drop box. The value you write can be one of the elements loaded into the drop box or any other value.

VARDROP_BOX WRITE *ctl_id,element\$[,ERR=stmtref]*

Update Current Setting. Use **VARDROP_BOX WRITE** with a string expression to update the current selection by element.

VARDROP_BOX WRITE *ctl_id,index[,ERR=stmtref]*

Update Current Index. Use **VARDROP_BOX WRITE** with a numeric expression to update the current selection by element index.

Format 15: *Clear Current Selection*

VARDROP_BOX WRITE *ctl_id*, "" [,ERR=*stmtref*]

Use this format to clear the currently selected entry in variable drop boxes.



Note: This behavior can be altered by use of the '+N' & '-N' Mnemonics, [p.623](#).

Format 16: *Report All Changes*

VARDROP_BOX AUTO *ctl_id* [,ERR=*stmtref*]

Use the **VARDROP_BOX AUTO** format to have ProvideX generate a CTL value whenever the current selection is changed. This lets you track changes in *which selection is highlighted* in a **VARDROP_BOX**.

VARLIST_BOX Directive

Control Variable List Box

Formats

1. *Define/Create*: **VARLIST_BOX** *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]
2. *Remove*: **VARLIST_BOX REMOVE** *ctl_id*[,ERR=*stmtref*]
3. *Disable/Enable*: **VARLIST_BOX** {**DISABLE** | **ENABLE**}*ctl_id*[,ERR=*stmtref*]
4. *Force Focus*: **VARLIST_BOX GOTO** *ctl_id*[,ERR=*stmtref*]
5. *Signal on Focus*: **VARLIST_BOX SET_FOCUS** *ctl_id*, *ctl_val*[,ERR=*stmtref*]
6. *Load List*: **VARLIST_BOX LOAD** *ctl_id*,*dln_list*\$[,ERR=*stmtref*]
7. *Load Array*: **VARLIST_BOX LOAD** *ctl_id*,*array_name*#{**ALL**}[,ERR=*stmtref*]
Note: The curly braces enclosing **{ALL}** are part of the syntax.
8. *Load/Delete Element*: **VARLIST_BOX LOAD** *ctl_id*,*index*,{*element*\$ | *}[,ERR=*stmtref*]
9. *Retrieve Element*: **VARLIST_BOX FIND** *ctl_id*,*index*,*var*\$[,ERR=*stmtref*]
10. *Read Current Selection*: **VARLIST_BOX READ** *ctl_id*,*var*\$[,*mode*\$][,ERR=*stmtref*]
11. *Read Current Index*: **VARLIST_BOX READ** *ctl_id*,*var*[,*mode*\$][,ERR=*stmtref*]
12. *Update Current Item*: **VARLIST_BOX WRITE** *ctl_id*,*element*\$[,ERR=*stmtref*]
13. *Update Current Index*: **VARLIST_BOX WRITE** *ctl_id*,*index*[,ERR=*stmtref*]
14. *Clear Current Selection*: **VARLIST_BOX WRITE** *ctl_id*,""[,ERR=*stmtref*]
15. *Report All Changes*: **VARLIST_BOX AUTO** *ctl_id*[,ERR=*stmtref*]

Where:

@(col,ln,wth,ht)	Position and size of the variable list box region when expanded. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines. (Note that list box height, when not expanded, is governed by the system and is roughly 1.5 times the standard graphic font height.)
<i>array_name</i> \$	Name of array to load into variable list box. String variable followed by {ALL} .
<i>ctl_id</i>	Unique logical identifier for a variable list box (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the F4 key) or Negative CTL Definitions, p.817 . Use this value with the apostrophe operator to access Variable List Box Properties .
<i>ctl_val</i>	CTL value to generate when the variable list box gains focus.
<i>ctrlopt</i>	Control options. Supported options for VARLIST_BOX include: ERR= <i>stmtref</i> Error transfer FNT= "font,size[,attr]" Font name, size, optional properties Refer to the 'FONT' Mnemonic, p.609 for details.

	KEY=char\$ Hot key
	LEN=num Maximum input length
	MSG=text\$ Message line
	MNU=ctl CTL value associated with right-click menu event.
	TIP=text\$ Mouse pointer message.
	To change the colour, refer to the ' TC '= System Parameter, p.689 .
	OWN=name\$ Name assigned for automated testing of this control.
	OPT=char\$ Attribute/behaviour settings:
	"D" - <i>Disabled</i> . User cannot access the list box.
	"G" - <i>Global</i> . Keep active on focus change to new/non-concurrent window.
	"H" - <i>Hide</i> . Do not display the list box.
	"S" - <i>Signal</i> . Generate CTL value but without shifting focus.
	"s" - <i>Scroll</i> . Allow scroll within resizable/scrollable dialogue box.
	Some characters may be combined. Invalid settings are ignored.
dln_list\$	Delimited list of elements to load. String expressions.
element\$	Single element to load. String expression. Use the <i>asterisk</i> * instead to <i>delete</i> an element. For instance, <code>VARLIST_BOX LOAD 86,4,*</code> will "eighty-six" (remove) element 4 from VARLIST_BOX 86.
index	Position of the element in the variable list box. Numeric expression. Integers: the index of the 1 st element is 1.
mode\$	String variable. ProvideX returns a single-character hex value in this variable to report the last method / keystroke the user chose to select an item from the list box (\$01\$ for MOUSE-CLICK or \$0D\$ for Enter).
stmtref	Program line number or label to transfer control to.
var[\$]	Variable to receive value. String variable for element/numeric for index.

Description Use the **VARLIST_BOX** directive to create and control variable list boxes on the screen. That is, the user can select any element from a list of items associated with the variable list box or can enter *any other value*.

Variable List Box Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a variable list box are described in [Chapter 7. Control Object Properties, p.708](#).

Format 1: Define/Create

```
VARLIST_BOX ctl_id,@(col,ln,wth,ht)[,ctrlopt]
```

Use the format above to create a variable list box, giving it a unique *ctl_id*. When a user selects an item from a variable list box, or enters an item that is not on the list, the associated *ctl_id* you give to the variable list box is used to generate a CTL value.

Use the **FNT=** option to establish the font for the variable list box. If you omit the font option, ProvideX uses the system default font. Use **FNT=""** to set the font as standard text mode fixed font.

The following example creates a variable list box that generates a CTL=100 when any item is selected from it. It's loaded with the items Cat, Dog, and Pig.

```
0010 VARLIST_BOX 100,@(2,14,12,6)
0020 VARLIST_BOX LOAD 100,"Cat/Dog/Pig/"
```

The user can select any of the three items supplied or enter any other value.

Format 2: *Remove*

```
VARLIST_BOX REMOVE ctl_id[,ERR=stmtref]
```

Use the **VARLIST_BOX REMOVE** format to delete a variable list box.

Format 3: *Disable/Enable*

```
VARLIST_BOX {DISABLE | ENABLE}ctl_id[,ERR=stmtref]
```

Use the **VARLIST_BOX DISABLE** format to gray out a variable list box so that it will be visible but *inaccessible* to users. To reactivate it, use **VARLIST_BOX ENABLE**.

Format 4: *Force Focus*

```
VARLIST_BOX GOTO ctl_id[,ERR=stmtref]
```

Use **VARLIST_BOX GOTO** to reactivate and force focus to a variable list box, ready for the next user action.

Format 5: *Signal on Focus*

```
VARLIST_BOX SET_FOCUS ctl_id,ctl_val[,ERR=stmtref]
```

Use the **VARLIST_BOX SET_FOCUS** format to define an alternate CTL value to generate whenever focus shifts to the variable list box.

Formats 6, 7 and 8: *Load a Variable List Box*

Use the **VARLIST_BOX LOAD** formats below to load items into a variable list box. The element(s) can be loaded as a *delimited string*, as an *array of string elements*, or *individually*.

```
VARLIST_BOX LOAD ctl_id,dlim_list$[,ERR=stmtref]
```

Load List. When you load items from a *delimited string*, the last character in the string must be the delimiter; e.g.,

```
0100 VARLIST_BOX LOAD 10000,"Fox/Cat/Dog/Cow/Sheep/Horse/Pig/Elephant/Ant/"
0500 VARLIST_BOX LOAD 15000,"Fox"+SEP+"Cat"+SEP+"Dog"+SEP
```

VARLIST_BOX LOAD *ctl_id,array_name* \${ALL}[,ERR=*stmtref*]

Load Array. Use this format to load a complete *array* into the variable list box. Note that the curly braces enclosing **{ALL}** are part of the syntax.

VARLIST_BOX LOAD *ctl_id,index*,{*element*\$ | *},[ERR=*stmtref*]

Load Element. When loading a variable list box one element at a time, the index value refers to the element before which the new element is to be inserted. For instance, if *index* is 1, the new element will be inserted before 1, at the start of the list. If *index* is 0 *zero*, the new element will be appended to the end of the list.

If you have more items on a list than will fit the physical screen size of a variable list box, ProvideX automatically supplies scrollbars. To delete or remove a specified element from a variable list box, use an *asterisk* * in place of the element string; e.g.,

VARLIST_BOX LOAD 86,4,* ! Deletes item whose index=4 from list box 86.

Format 9: Retrieve Element

VARLIST_BOX FIND *ctl_id,index,var* \$[,ERR=*stmtref*]

Use **VARLIST_BOX FIND** to retrieve a specific element from a list box.

Formats 10 and 11: Read Current Selection

Use the **VARLIST_BOX READ** formats to read which element the user has selected from the variable list box. Note that you must read the user's selection before an application can use it.

Use a string variable to return information on how the element was selected. The value returned will be:

\$02\$ for **DOUBLE MOUSE-CLICK**.

\$0D\$ for **Enter**.

After this value is read, it resets to \$00\$ (null).

VARLIST_BOX READ *ctl_id,var* \$[,*mode*\$][,ERR=*stmtref*]

Read Current Element. Use a string variable in a **VARLIST_BOX READ** to receive the value of the currently selected element. Use an optional second variable to receive the selection method (i.e., *mode*\$).

VARLIST_BOX READ *ctl_id,var* [,*mode*\$][,ERR=*stmtref*]

Read Current Index. Use a numeric variable in a **VARLIST_BOX READ** to receive the index of the element the user has selected. Once a user has made a selection, it is your responsibility to read it to return the value to your program.

Formats 12 and 13: *Write Current Selection*

Use the **VARLIST_BOX WRITE** formats described below to update the current selection in the variable list box. The value you write can be one of the elements loaded into the list box or any other value.

VARLIST_BOX WRITE *ctl_id,element\$* [,ERR=*stmtref*]

Update Current Setting. Use **VARLIST_BOX WRITE** with a string expression to update the current selection by element.

VARLIST_BOX WRITE *ctl_id,index* [,ERR=*stmtref*]

Update Current Index. Use **VARLIST_BOX WRITE** with a numeric expression to update the current selection by element index.

Format 14: *Clear Current Selection*

VARLIST_BOX WRITE *ctl_id,""* [,ERR=*stmtref*]

Use this format to clear the currently selected entry in variable list boxes.



Note: This behavior can be altered by use of the '+N' & '-N' Mnemonics, p.623.

Format 15: *Report All Changes*

VARLIST_BOX AUTO *ctl_id* [,ERR=*stmtref*]

Use the **VARLIST_BOX AUTO** format to have ProvideX generate a CTL value whenever the current selection is changed. This lets you track changes in *which selection is highlighted* in a **VARLIST_BOX**.

V_SCROLLBAR Directive

Control Vertical Scrollbar

Formats

1. *Define/Create*: **V_SCROLLBAR** *ctl_id*,@(*col*,*ln*,*wth*,*ht*)[*ctrlopt*]
2. *Define at Edge of Window*: **V_SCROLLBAR** *ctl_id* WINDOW [*ctrlopt*]
3. *Remove*: **V_SCROLLBAR REMOVE** *ctl_id*[*ERR=stmtref*]
4. *Disable/Enable*: **V_SCROLLBAR** {ENABLE | DISABLE} *ctl_id*[*ERR=stmtref*]
5. *Hide/Show*: **V_SCROLLBAR** {HIDE | SHOW} *ctl_id*[*ERR=stmtref*]
6. *Force Focus*: **V_SCROLLBAR GOTO** *ctl_id*[*ERR=stmtref*]
7. *Read*: **V_SCROLLBAR READ** *ctl_id*,*setting*,*max*,[*rgn_chg*][*arrow_chg*][*ERR=stmtref*]
8. *Update*: **V_SCROLLBAR WRITE** *ctl_id*,*marker*,*max*[*ERR=stmtref*]

Where

- @(col,ln,wth,ht)** Position and size of the vertical scrollbar region. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.
- arrow_chg** Amount to increase/decrease the **V_SCROLLBAR** setting when the user selects the *arrow* at the top/bottom edge of the vertical scrollbar. Numeric expression. (Default: 1)
- ctl_id** Unique logical identifier for a vertical scrollbar (any integer -32000 to +32000). Avoid integers that conflict with keyboard definitions (e.g., 4 cancels CTL=4 for the **F4** key) or **Negative CTL Definitions, p.817**. Use this value with the apostrophe operator to access various **Variable List Box Properties**.
- ctrlopt** Control options. Supported options for **V_SCROLLBAR** include:
ERR=stmtref Error transfer
OWN=name\$ Name assigned for automated testing of this control.
OPT=char\$ Attribute/behaviour settings:
 "D" - *Disabled*. User cannot access the scrollbar.
 "G" - *Global*. Keep active on focus change to new/non-concurrent window.
 "H" - *Hide*. Do not display the scroll bar.
 "A" - *Auto*. Generate CTL signal for each movement.
 "S" - *Scroll*. Allow scroll within resizable/scrollable dialogue box.
 Some characters may be combined. Invalid settings are ignored.
- marker** Relative position to set as the **V_SCROLLBAR** *marker*. Numeric expression between 1 and the maximum.
- max** Logical maximum value of the **V_SCROLLBAR**. Numeric expression.
- rgn_chg** Amount to increase/decrease the **V_SCROLLBAR** setting when the user selects the *region* above/below the *marker*. Numeric expression. (Default: max/width.)
- setting** Numeric variable to receive the current scrollbar setting.

stmtref Program line number or label to transfer control to.

Description Use the **V_SCROLLBAR** directive to create a vertical scrollbar control object on the screen. Your program logic can read and adjust a value by increments to control logical record position within a file every time the user moves the vertical scrollbar.

Use either **V_SCROLLBAR** Format 1 or 2 (described below) to define or create a vertical scrollbar. The value in *ctl_id* gives the vertical scrollbar a unique identifier. This is generated as a CTL value whenever the vertical scrollbar is selected and changed.

Vertical Scrollbar Properties

The **Apostrophe Operator** can be used with the unique logical identifier (*ctl_id*) to dynamically read and alter a wide variety of control attributes (properties) directly from the programming language. The list of properties available for manipulating a vertical scrollbar are described in [Chapter 7. Control Object Properties, p.708](#).

Format 1: Create

V_SCROLLBAR *ctl_id*,@(col,ln,wth,ht)[,ctrlopt]

Use this format to create a vertical scrollbar inside the current window; e.g.,

```
0010 V_SCROLLBAR 100 ,@(70 , 2 , 2 , 20)
```

defines a vertical scrollbar 2 columns *wide*, 20 lines *high*, starting at *column 70 on line 2*. Whenever the scrollbar is selected a CTL=100 is generated.

Format 2: Define at Edge of Window

V_SCROLLBAR *ctl_id* WINDOW[,ctrlopt]

Use the **V_SCROLLBAR** format with **WINDOW** to create a vertical scrollbar at the edge of the window.

Format 3: Remove

V_SCROLLBAR REMOVE *ctl_id*[,ERR=*stmtref*]

Use the **V_SCROLLBAR REMOVE** format to delete the vertical scrollbar.

Format 4: Disable/Enable

V_SCROLLBAR {DISABLE | ENABLE} *ctl_id* [,ERR=*stmtref*]

Use the **V_SCROLLBAR DISABLE** format to gray-out a vertical scrollbar so that it will be visible but *inaccessible* to users. To reactivate it, use **V_SCROLLBAR ENABLE**.

Format 5: Hide/Show

V_SCROLLBAR {HIDE | SHOW} *ctl_id*[,ERR=*stmtref*]

With the **V_SCROLLBAR HIDE** format, the scrollbar remains active, but is not displayed. It is still accessible programmatically. Use the **SHOW** format to restore the display and user access.

Format 6: Force Focus

V_SCROLLBAR GOTO *ctl_id* [,ERR=*stmtref*]

Use the **V_SCROLLBAR GOTO** format to reactivate and force focus to a vertical scrollbar, ready for the next user action.

Format 7: Read Setting

V_SCROLLBAR READ *ctl_id,setting,max,[rgn_chg]* [,*arrow_chg*] [,ERR=*stmtref*]

Use this format to read the current setting of the vertical scrollbar.



Note: Once a new position is selected, you must read it before your application can use the value to update the actual **V_SCROLLBAR** position.

Example:

```
0120 V_SCROLLBAR READ 100,X,1000
0130 V_SCROLLBAR WRITE 100,X,1000
```

Line 0120 reads the scrollbar position relative to 1000 and line 0130 updates the settings.

Format 8: Update

V_SCROLLBAR WRITE *ctl_id,marker,max* [,ERR=*stmtref*]

Use the **V_SCROLLBAR WRITE** format to update or write the **V_SCROLLBAR** settings.

See Also

[H_SCROLLBAR Control Horizontal Scroll Bar, p.153](#),
[Chapter 7. Control Object Properties, p.701](#).

Examples

```
0110 LET VAL=1,MX=400,BJMP=25,SJMP=1

0120 V_SCROLLBAR 101,@(6,14,1,10)
0130 V_SCROLLBAR 102,@(15,14,2,10)
0140 V_SCROLLBAR 103,@(25,14,3,10)
0150 V_SCROLLBAR 104 WINDOW
0160 INPUT (0,HLP="V_SCROLLBAR")@(40,18),"Select...: ", 'CL' ,X$
0170 IF CTL<101 OR CTL>104 THEN GOTO 0210
0180 V_SCROLLBAR READ CTL,VAL,MX,BJMP,SJMP
0190 V_SCROLLBAR WRITE CTL,VAL,MX
0200 PRINT @(40,19),"Selection:",CTL,":",STR(VAL),'CL',; GOTO 0160
0210 IF CTL=0 OR CTL>=3 THEN STOP ELSE GOTO 0160
0100 ! 100 - Vertical Scroll Bar Example
```

VIA Directive

Assign Variable Indirectly

Formats

1. *Assign to Variable*: **VIA** *var\$=expression* [,*var\$=...*] [,**ERR=stmtref**]
 2. *Assign to Composite Variable*: **VIA** *comp\$,var_1\$[,subscr]=expression[,...n]* [,**ERR=stmtref**]
- Note*: The inner set of brackets enclosing [,*subscr*] are part of the syntax.

Where:

- comp\$* Composite string variable. (Name of variable defined as composite string.).
- [,*subscr*] Optional subscript(s) of a variable in the composite string. String expression. You can include from 1 to 3 optional numeric expressions in square brackets, comma-separated, as subscripts for the variable.
- expression* Value you want assigned to the named variable.
- var\$* String variable containing the name of the variable to be used.
- var_1\$* to *var_2\$* String expression containing the name of a variable in the composite string.
- stmtref* Program line number or label to transfer control to.

Description

Use the **VIA** directive to assign a value to a variable where the variable name is contained in another string variable. The target variable's type (numeric or string) is based on the type of expression.

Format 1: *Assign to Variable*

VIA *var\$=expression* [,*var\$=...*]

Use this **VIA** format to assign the value in the expression to a variable whose name is stored in *var\$* (i.e., to assign **VIA** the *var\$*). If you use a string, then ProvideX automatically appends a \$ to the variable name.

Example:

```
0030 LET ANIMAL$="pet ", PET$="dog"
0040 VIA ANIMAL$="cat "
0050 PRINT ANIMAL$, " ",PET$
```

... when run, the result is pet cat.

```
0060 LET VAR$="number ", NUMBER=23
0070 VIA VAR$=30
0080 PRINT VAR$, " ",NUMBER; END
```

... when run, the result is number 30.

Format 2: *Assign to Composite Variable*

VIA *composite\$,var_1\$*[[*,subscr*]]=*expression*[*,...n*]

You can assign the value of the expression to a variable in a composite string. In this format, you can specify up to 3 subscripts for the variable(s).

Example:

```
0010 IOLIST X$,Y$,Z$
0020 DIM PET$:IOL=0010
0030 LET X$="Dogs",Y$="Cats",Z$="!" REM Assign values to variables
0040 VIA PET$,"X$"="Lotsa",PET$,"Y$"="and",PET$,"Z$"="It's raining"
0050 PRINT PET.X$," ",X$," ",PET.Y$," ",Y$
0060 LET PET.X$=X$,PET.Y$=Y$
0070 PRINT PET.Z$," ",PET.Y$," and ",PET.X$,Z$
-:run
Lotsa Dogs and Cats
It's raining Cats and Dogs!
```

VIDEO_PALETTE Directive Control Video Colours

- Formats
1. *Read Palette*: `VIDEO_PALETTE READ var$[,ERR=stmtref]`
 2. *Change Palette*: `VIDEO_PALETTE string$[,ERR=stmtref]`
 3. *Read Palette Indexed Table*: `VIDEO_PALETTE INDEXED READ var$[,ERR=stmtref]`
 4. *Change Palette Indexed Table*: `VIDEO_PALETTE INDEXED string$[,ERR=stmtref]`

Where

- string\$* String expression containing the new **VIDEO_PALETTE** or **INDEXED** table settings to be used.
- stmtref* Program line number or label to transfer control to.
- var\$* String variable to receive the current **VIDEO_PALETTE** or **INDEXED** table settings.

Description ProvideX supplies access to the *standard* 0 - 15 colours as well as any *extended* 16 - 254 colours in the colour video palette (internal colour index) in Windows. This directive can be used to modify the palette in ProvideX; however, the '**OPTION**' mnemonic is the preferred method for assigning colours (by RGB code).

See Also '**COLOUR**' & '**_COLOUR**' Mnemonics, p.596
'**OPTION**' Mnemonic, p.624

Format 1: *Read Palette*

`VIDEO_PALETTE READ var$[,ERR=stmtref]`

Use this format to read the current settings of the video palette. The video palette consists of a string of 16 to 254 3-byte entries, where each byte defines the intensity of the colours Red, Green and Blue. For example, the 3-byte entry for bright (foreground) green is `$00FF00$`, magenta is `$FF00FF$` ...

`VIDEO_PALETTE READ X$` will return you the following defaults in a single string:

```
$000000 FF0000 00FF00 FFFF00$
$0000FF FF00FF 00FFFF FFFFFF$
$808080 800000 008000 808000$
$000080 800080 008080 C0C0C0$
```

The string is only punctuated by spaces to show you the initial sixteen 3-byte entries.

Format 2: *Change Palette*

VIDEO_PALETTE *string*\$[,ERR=*stmtref*]

Use this format to change/reset the video palette, as in **VIDEO_PALETTE X\$** where *X\$* contains your new values.



Note: Changing the palette may not work on all systems. This depends on the colour capabilities of the PC hardware.

Format 3: *Read Palette Indexed Table*

VIDEO_PALETTE INDEXED READ *var*\$[,ERR=*stmtref*]

There is also an indexed table to reference the video palette. This index table consists of 48 entries: six 8-byte tables, where each byte represents an index into the video palette described above:

- The first 8 bytes represent the standard foreground colours.
- The second 8 bytes represent the colours used when the '**SB**' mnemonic is in effect
- The third 8 bytes represent the colours used when the '**BB**' mnemonic is in effect
- The fourth 8 bytes represent the colours used when both '**BB**' and '**SB**' are in effect
- The fifth 8 bytes represent the "Background" colours
- The sixth 8 bytes represent the colours used when '**BR**' and '**SB**' are in effect

Use the **READ** format to return the current settings in the index table, as in **VIDEO_PALETTE INDEXED READ X\$**. For more information, refer to the '**BB**' Mnemonic, [p.588](#), the '**BR**' Mnemonic, [p.592](#), and the '**SB**' Mnemonic, [p.638](#).

Format 4: *Change Palette Indexed Table*

VIDEO_PALETTE INDEXED *string*\$[,ERR=*stmtref*]

Use this format to change/reset the video palette index table, as in **VIDEO_PALETTE INDEXED X\$** where *X\$* contains your new values.

WAIT Directive

Temporarily Halt Execution

Format **WAIT** *seconds*

Where:

seconds Value determines the number of seconds to wait. Numeric expression.

Description Use the **WAIT** directive to have the system suspend execution for the number of seconds you indicate in the numeric expression. During the time the program is suspended, the **SETESC** directive is deferred. This directive will only be processed once the **WAIT** interval has elapsed.



Note: The accuracy of the timing varies from operating system to operating system. On most UNIX/Linux systems, the **WAIT** is accurate to within one second.

Example

```
0110 READ (1,ERR=1000,KEY=K$)N$
1000 REM Test for busy record
1010 IF ERR=0 THEN PRINT @(0,22),"Busy. will retry";
1010: WAIT 5; RETRY
```



Note: **WAIT 0** is often used to flush the display cache; e.g.,
PRINT 'IMAGE'("Bar"), 'FILL'(1,4), 'PEN'(1,1,4), 'IMAGE'("),
WAIT 0

WAIT FOR EVENT Directive

Wait for Event

Format

WAIT FOR EVENT



Note: For more information on internal events and event handling in ProvideX, refer to the directives [ENABLE EVENT Internal Event Enable, p.112](#), and [DISABLE Disable Use of Prefix Table Entry, p.92](#).

Description

Use the **WAIT FOR EVENT** directive to relinquish control and wait until an event occurs from any object.

See Also

[ENABLE EVENT Directive, p.112](#)
[DISABLE EVENT Directive, p.94](#)

WEND Directive

End WHILE Loop

Format **WEND**



Note: Refer to [WHILE..WEND Repeat Statements, p.375](#), for complete syntax.

Description The **WEND** directive marks the end of a **WHILE/WEND** loop. When ProvideX reaches the **WEND** directive, the expression defined in the currently active **WHILE** is re-evaluated. If the result of the evaluation is not zero, control is returned to the directive following the **WHILE**. If the value is 0 (zero, false), control transfers to the directive following the **WEND**.

See Also [WHILE..WEND Repeat Statements, p.375](#)

Example

```
0100 WHILE A$<>"E"
0110 LET K$=KEY(1,END=1000)
0120 READ (1)N$
0130 PRINT "Key: ",K$," Name: ",N$
0140 INPUT "Delete (Y/N):",A$
0150 IF A$="Y" THEN REMOVE (1,KEY=K$)
0160 WEND
0170 STOP
1000 PRINT "End-of-file..."
1010 EXITTO 0170
```

WHILE..WEND Directive

Repeat Statements

Format **WHILE** *expression* ..**WEND**

Where:

expression Numeric expression. When the evaluated value of the expression is 0 (zero, false) looping ends.

WEND Directive required to end the **WHILE** sequence.

Description Use the **WHILE** directive for conditional looping in a program. ProvideX executes all directives between a **WHILE** directive and the next **WEND** directive repeatedly until the value of the *expression* is 0 zero.

When ProvideX encounters a **WHILE** directive, it evaluates the *expression*. If the result is not 0 zero, ProvideX continues execution until a corresponding **WEND** directive is encountered, at which point the *expression* is re-evaluated. ProvideX continues to loop back to the directive following the **WHILE** directive until a 0 value is reached. At this point, ProvideX advances to the next **WEND** directive, where it terminates the loop. Then control transfers to the statement following the **WEND**.

The *expression* would normally include a logical operator (such as an *equals =*, *less-than* symbol <, or the **LIKE Operator**, p.822), but you can use any numeric expression.

If ProvideX encounters a **WEND** directive but is not currently processing a **WHILE / WEND** loop, it returns an Error #27: Unexpected or incorrect WEND, RETURN, or NEXT. All **FOR/NEXT**, **GOSUB/RETURN**, and **WHILE/WEND** sequences executed within the **WHILE/WEND** loop must be completed.

Use a conditional **EXITTO** or **BREAK** to exit a **WHILE/WEND** loop early.

See Also **BREAK Immediate Exit of Loop**, p.33,
CONTINUE Initiates Next Iteration of Loop, p.57
EXITTO End Loop, Transfer Control, p.125
LIKE Operator, p.822
Loop Structures, *User's Guide*.

Example

```
0100 WHILE A$<>"E"
0110 LET K$=KEY(1,END=1000)
0120 READ (1)N$
0130 PRINT "Key: ",K$," Name: ",N$
0140 INPUT "Delete (Y/N):",A$
0150 IF A$="Y" THEN REMOVE (1,KEY=K$)
0160 WEND
0170 STOP
1000 PRINT "End-of-file..."
1010 EXITTO 0170
```

WINPRT_SETUP Directive Windows Printer Setup

Formats

1. *Read Current Selection*: WINPRT_SETUP [SERVER] READ var\$[,ERR=stmtref]
2. *Read Properties*: WINPRT_SETUP [SERVER] READ PROPERTIES var\$[,ERR=stmtref]
3. *Update Selection*: WINPRT_SETUP [SERVER] WRITE printer\$[,ERR=stmtref]
4. *Update Properties*: WINPRT_SETUP [SERVER] WRITE PROPERTIES settings\$[,ERR=stmtref]
5. *List Available Printers*: WINPRT_SETUP [SERVER] LIST var\$[,ERR=stmtref]
6. *List Printer Names Only*: WINPRT_SETUP [SERVER] DIRECTORY var\$
7. *Display Printer Dialogue*: WINPRT_SETUP INPUT var\$[,ERR=stmtref]

Where

- printer\$* String expression to select the current printer by name.
- settings\$* String expression for assigning property values.
- SERVER** Optional keyword enabling access to printers defined on a Windows server when using WindX.
- stmtref* Program line number or label to transfer control to.
- var\$* String variable for the returned value for printer name(s) / properties.

Description

Use **WINPRT_SETUP** to control settings for the currently-selected Windows printer and its properties. This directive can also be used to access and report information about the printer's available on the system.

If network account privileges are a consideration, set the '**AW**' system parameter to ensure that **WINPRT_SETUP** accesses printers that are available to the current user. Otherwise, information about all printers may be reported.

WINPRT_SETUP Properties

The default properties of a printer are determined by the individual printer/driver manufacturers. Some properties are not available for all drivers (e.g., collate, ordered printing). Others vary from printer to printer (e.g., font, point size). Default paper size is usually letter size (8 1/2" x 11", expressed as PAPER_SIZE=1 for DMPAPER_LETTER). Refer to [Paper Sizes](#) below for the complete list of **PAPER_SIZE=num** codes.

The following properties can be assigned to a Windows printer:

- | | |
|---------------------------------|--|
| COLLATE= YES NO AUTO | Collate paper (if supported by printer driver). |
| COLOUR= YES NO | Coloured print (if supported by printer driver). |
| COPIES= num | Set number of copies. |
| DUPLEX= 1 2 3 | Set for double-sided printing (if supported by printer driver) where |
| | 1 = Duplex is off |
| | 2 = Duplex in Portrait |
| | 3 = Duplex in Landscape. |

FILE= + - <i>filename</i>	Print to file. Use a - <i>minus</i> to disable prompt and print directly to <i>filename</i> .
FORCE6X10= YES NO	Adjust column width to 60% of the line height as defined in font size specifications. This may solve minor alignment issues when printing proportional fonts to a graphical print device.
MARGINS= <i>left:top:right:bottom</i>	Define margins. See Margin Control below.
OFFSET= <i>x:y</i>	Offset print area from the upper left corner of the page. Values <i>x</i> and <i>y</i> are in <i>thousandths of an inch</i> (e.g., OFFSET=750:500 sets the print area three-quarters of an inch from the left margin and half an inch down from the top.)
ORIENTATION= PORTRAIT LANDSCAPE	Swaps output width for length, and vice versa. See also *WINPRT* Printing Options, p.629 .
PAPERLENGTH= <i>num</i>	Force specific paper length in 1/10 millimetres.
PAPERSIZE= <i>num</i>	Define paper size. See Paper Sizes below.
PAPERWIDTH= <i>num</i>	Force specific paper width in 1/10 millimetres.
PRESTRETCH	Override default image stretching.
QUALITY= -1 -2 -3 -4	Specifies print density, where -1 = Draft -2 = Low Resolution -3 = Medium Resolution -4 = High Resolution
RANGE= <i>from:to</i>	Select page range to print.
RESOLUTION= <i>x:y</i>	Set specific resolution by defining <i>x:y</i> in dots per inch (DPI). This value overrides the QUALITY= parameter.
SCALE= <i>num</i>	Reduce or increase the size of output image as a percentage of the original.
SOURCE= <i>num</i>	Set specific Paper Source (below). See also *WINPRT* Printing Options, p.629 .
TRUETYPE= <i>num</i>	Set True Type font handling where 1 = Print TT fonts as graphics 2 = Download TT fonts as soft fonts 3 = Substitute device fonts for TT fonts

Margin Control

The **MARGINS=** property accepts values for *left*, *top*, *right* and *bottom* in 1000^{ths} of an inch. These values affect both text mode printing (e.g., `PRINT (x) "ABC",`) and graphical printing (e.g., `PRINT (x) 'PICTURE(. . .),`). The following example indicates a 1 inch margin on all sides of the page.

```
WINPRT_SETUP WRITE PROPERTIES "MARGINS=1000:1000:1000:1000"
```

The default values are -1 : -1 : -1 : -1, which is the equivalent of no margin setting. If only two values are given, the *right* and *bottom* margins default to the hardware-imposed print margins. A value of -1 indicates that the printer is to use its default physical margin.

The software internally adjusts the top and left margins by any hardware imposed printing offsets. Most laser or ink jet printers impose a margin around the edge of the paper where they cannot print. These values are taken into account by the property settings in order to assure consistent output positioning regardless of printer type.

Note: Due to printer wear and tear, paper slippage, and other outside factors, the output position is never guaranteed to be 100% correct. However, the ProvideX logic for handling margins is consistent with most other Windows-based software. The alignment will be similar to the output from other programs printed on the same printer using the same OS and printer drivers.



The keyword **MARGIN** may be used instead of **MARGINS**, if preferred; however, **WINPRT_SETUP READ PROPERTIES** always returns **MARGINS=** as the property name.

Paper Sizes

The **PAPERSIZE=num** option is mapped to paper sizes according to *print.h*. The following chart shows the *num* value followed by the internal name and a general description of the paper.



Warning: The PostScript driver mistakenly uses **DMPAPER_** values between * 50 and 56. Do not use this range when defining new paper sizes.

<i>num</i>	<i>Internal Name</i>	<i>Paper Size / Description</i>
1	DMPAPER_LETTER	Letter 8 1/2 x 11 in
2	DMPAPER_LETTERS <small>SMALL</small>	Letter Small 8 1/2 x 11 in
3	DMPAPER_TABLOID	Tabloid 11 x 17 in
4	DMPAPER_LEDGER	Ledger 17 x 11 in
5	DMPAPER_LEGAL	Legal 8 1/2 x 14 in
6	DMPAPER_STATEMENT	Statement 5 1/2 x 8 1/2 in
7	DMPAPER_EXECUTIVE	Executive 7 1/4 x 10 1/2 in
8	DMPAPER_A3	A3 297 x 420 mm
9	DMPAPER_A4	A4 210 x 297 mm
10	DMPAPER_A4 <small>SMALL</small>	A4 Small 210 x 297 mm
11	DMPAPER_A5	A5 148 x 210 mm
12	DMPAPER_B4	B4 250 x 354
13	DMPAPER_B5	B5 182 x 257 mm
14	DMPAPER_FOLIO	Folio 8 1/2 x 13 in
15	DMPAPER_QUARTO	Quarto 215 x 275 mm
16	DMPAPER_10x14	10x14 in
17	DMPAPER_11x17	11x17 in
18	DMPAPER_NOTE	Note 8 1/2 x 11 in
19	DMPAPER_ENV_9	Envelope #9 3 7/8 x 8 7/8
20	DMPAPER_ENV_10	Envelope #10 4 1/8 x 9 1/2

<i>num</i>	<i>Internal Name</i>	<i>Paper Size / Description</i>
21	DMPAPER_ENV_11	Envelope #11 4 1/2 x 10 3/8
22	DMPAPER_ENV_12	Envelope #12 4 1/2 x 11
23	DMPAPER_ENV_14	Envelope #14 5 x 11 1/2
24	DMPAPER_CSHEET	C size sheet
25	DMPAPER_DSHEET	D size sheet
26	DMPAPER_ESHEET	E size sheet
27	DMPAPER_ENV_DL	Envelope DL 110 x 220mm
28	DMPAPER_ENV_C5	Envelope C5 162 x 229 mm
29	DMPAPER_ENV_C3	Envelope C3 324 x 458 mm
30	DMPAPER_ENV_C4	Envelope C4 229 x 324 mm
31	DMPAPER_ENV_C6	Envelope C6 114 x 162 mm
32	DMPAPER_ENV_C65	Envelope C65 114 x 229 mm
33	DMPAPER_ENV_B4	Envelope B4 250 x 353 mm
34	DMPAPER_ENV_B5	Envelope B5 176 x 250 mm
35	DMPAPER_ENV_B6	Envelope B6 176 x 125 mm
36	DMPAPER_ENV_ITALY	Envelope 110 x 230 mm
37	DMPAPER_ENV_MONARCH	Envelope Monarch 3.875 x 7.5 in
38	DMPAPER_ENV_PERSONAL	6 3/4 Envelope 3 5/8 x 6 1/2 in
39	DMPAPER_FANFOLD_US	US Std Fanfold 14 7/8 x 11 in
40	DMPAPER_FANFOLD_STD_GERMAN	German Std Fanfold 8 1/2 x 12 in
41	DMPAPER_FANFOLD_LGL_GERMAN	German Legal Fanfold 8 1/2 x 13 in
256	DMPAPER_USER	

Paper Source

The `SOURCE=num` option is mapped to the following internal paper tray definitions:

<i>num</i>	<i>Internal Name</i>	<i>num</i>	<i>Internal Name</i>
1	DMBIN_UPPER	6	DMBIN_ENVMANUAL
1	DMBIN_ONLYONE	7	DMBIN_AUTO
2	DMBIN_LOWER	8	DMBIN_TRACTOR
3	DMBIN_MIDDLE	9	DMBIN_SMALLFMT
4	DMBIN_MANUAL	10	DMBIN_LARGEFORMAT
5	DMBIN_ENVELOPE	14	DMBIN_CASSETTE

Format 1: Read Selection

WINPRT_SETUP READ *var\$* [*ERR=stmtref*]

Use the **WINPRT_SETUP READ** format to read the currently selected default printer name. The value returned in *var\$* will be the OS name of an existing physical queue.

Example:

```
-: WINPRT_SETUP READ PRTR$
-: ?PRTR$
HPLaserJet III on \\machine_1\hp
```

Format 2: Read Properties

WINPRT_SETUP READ PROPERTIES *var\$*[,ERR=*stmtref*]

Use the **WINPRT_SETUP READ PROPERTIES** format to read the current settings of all properties associated with your current printer. The values are returned in a semicolon-separated list.

Example:

```
->OPEN (1) "*WINPRT*"
->WINPRT_SETUP READ PROPERTIES PROP$
->?PROP$
RANGE=ALL;COLLATE=NO;COPIES=1;ORIENTATION=PORTRAIT;PAPERSIZE=1;SOURCE=1;RE
SOLUTION=300:300;OFFSET=0:0;TRUETYPE=2;DRIVER=WINSPOOL
```

Format 3: Update Selection

WINPRT_SETUP WRITE *printer\$*[,ERR=*stmtref*]

Use the **WINPRT_SETUP WRITE** format to write or change the currently selected default printer, using a string expression to pass the system the name of the printer to use.

Example:

```
->WINPRT_SETUP WRITE "Canon BJC-4300 on \\machine_2\canon"
```

Format 4: Update Properties

WINPRT_SETUP WRITE PROPERTIES *settings\$*[,ERR=*stmtref*]

Use the **WINPRT_SETUP WRITE PROPERTIES** format to write or change the current properties, using a string expression to pass the system the property assignments to use.

Example:

```
LET P$="COPIES=2;ORIENTATION=LANDSCAPE"
WINPRT_SETUP WRITE PROPERTIES P$
```

Format 5: List Available Printers

WINPRT_SETUP LIST *var\$*[,ERR=*stmtref*]

Use **WINPRT_SETUP LIST** to obtain a comma-delimited list of all the printers defined in the system. Note that if you use this format on a workstation which does not have printers installed, there will be no value in your string variable and ProvideX returns an Error #12: File does not exist (or already exists).

Example:

```
->winprt_setup list choices$
->? choices$
HP LaserJet III on \\machine_1\hp,Envoy 7 Driver on EVY:,Canon BJC-4300 on
\\machine_2\canon,Acrobat PDFWriter on LPT1:,Default PostScript Printer on
LPT1:,Default PostScript Pri (Copy 1) on LPT1:,Acrobat Distiller 3.0 on
LPT1:,Acrobat Distiller 3.0 (Copy 1) on LPT1:,
```

Format 6: *List Printer Names Only*

WINPRT_SETUP DIRECTORY *var*\$

Using the **DIRECTORY** keyword returns a comma-delimited list of printers with just the printer portion of the names, excluding the ON Device portion.

Example:

```
->winprt_setup directory choices$
->? choices$
HP LaserJet III,Envoy 7 Driver,Canon BJC-4300,Acrobat PDFWriter,Default
PostScript Printer,Default PostScript Pri (Copy 1),Acrobat Distiller 3.0,
Acrobat Distiller 3.0 (Copy 1),HP LaserJet 2100 Series PCL 6,
```



Note: When the '**AW**' system parameter is set, the printer device name and port location returned by **WINPRT_SETUP DIRECTORY** may be presented differently.

Format 7: *Display Printer Dialogue*

WINPRT_SETUP INPUT *var*\$[,**ERR=stmtref**]

Use the **WINPRT_SETUP INPUT** format to display the Windows Printer Selection dialogue box, allowing the user to select a printer. The string variable returns the name of the printer selected by the user.

Example:

```
->? current$ ! Colour inkjet is current setting
Canon BJC-4300 on \\machine_1\canon
->winprt_setup input current$ ! User changes current selection to laser
->? current$
HP LaserJet 2100 Series PCL 6 on \\machine_1\hp
->
```

See Also

WINPRT [Windows Printing, p.760](#),
WINDEV [Raw Print Mode, p.756](#),
[Printing in Windows, User's Guide](#)
PDF [PDF Print Interface, p.744](#)

WITH..END WITH Directive Object Reference Construct

Format **WITH** *object* ..**END WITH**

Where:

END WITH Directive to end **WITH** construct.

object Object handle.

Description The **WITH** directive is used in *Object Oriented Programming* (OOP) to simplify the coding of multiple statements that refer to the same *object*. A logical "." variable is used in place of the object name prior to the **Apostrophe Operator** in all property/ method references; e.g.,

```
WITH Button_1.ct1
.'col=1,.'line=49,.'text$="Push Me"
END WITH
```

When a **WITH** directive is encountered, the current value of the logical "." variable is preserved on a stack, which is restored upon execution of an **END WITH**. Each **WITH** should be terminated by an **END WITH**. The "." variable is only allowed to be referenced as an object handle; therefore, any other "." references (without the **Apostrophe Operator**) are invalid; e.g.,

```
. = 3           (Invalid)
PRINT .        (Invalid)
.'value$="ABC" (Valid)
```

The value of the "." variable is *global*; i.e., if it is set in mainline code, it will be maintained over a **CALL** or **PERFORM** to a subprogram or object method. However, if it changes, the change *will not* be passed back to the mainline. Subroutines (**GOSUB**) can change the value and alter the **WITH** stack.

The **WITH** stack is maintained separately from the **GOSUB/FOR/WHILE** stack. Each program level (**CALL/PERFORM**) has its own **WITH** stack, which is freed upon exit of the program level. The maximum number of **WITH** values that can be stacked is 20 per program level. Attempting to issue an **END WITH** without a corresponding **WITH** will generate an Error #27 (Unexpected WEND, RETURN, or NEXT).

Transferring into the middle of a **WITH** structure is allowed; however, it is the developer's responsibility to assure that the **WITH** stack is properly maintained.

The current value of "." is available in **TCB(93)**.

See Also **Object Oriented Programming, p.22**

WRITE Directive

Add/Update Data in File

Formats

1. *Write*: **WRITE** (*chan*[,*fileopt*])*varlist*
2. *Write Lock*: **WRITE LOCK** (*chan*[,*fileopt*])*varlist*

Where:

<i>chan</i>	Channel or logical file number of the file to which to write.
<i>fileopt</i>	Supported file options (see also, File Options, p.810): BSY=stmtref Traps Error #0: Record/file busy DOM=stmtref Missing record transfer END=stmtref END-OF-FILE transfer ERR=stmtref Error transfer IND=num Record index KEY=num Record key (see Automatic Padding with KEY=Option) REC=name\$ Record prefix (REC=VIS(string\$) can also be used) RTY=num Number of retries (one second intervals) TIM=num Maximum time-out (support write operations for TCP channels)
<i>stmtref</i>	Program line number or label to transfer control to.
<i>varlist</i>	Comma-separated list of variables, literals, and IOL= options.

Description

Use the **WRITE** directive to add/update a record to a file (logical file number / channel). ProvideX also supports use of the **WRITE** directive with ***MEMORY*** (a memory-resident file or queue of records).

Automatic Padding with KEY=Option

When you use **KEY=string\$:string\$[:string\$][...]** ProvideX automatically pads key segments. This is valid only if you have Keyed files with segmented key definitions. ProvideX right-pads the key segment using \$00\$ (nulls) to the segment's full length. The last segment in a compound key is not padded; e.g.,

```
KEYED "TEST", [1:1:5]+[2:1:6]+[3:1:8]
READ (1,KEY=A$:B$:C$)
```

is the same as

```
READ (1,KEY=PAD(A$,5,$00$)+PAD(B$,6,$00$)+C$)
```

Format 1: Write

WRITE (*chan*[,*fileopt*])*varlist*

If the specific record already exists (indexed, direct, or sort files) and you include the **DOM=stmtref** option, control transfers to the *stmtref*. Otherwise, the specified record is updated.

Examples:

```
0410 WRITE (1,ERR=1000,DOM=1200)A,B,Z9$
```

An **IND=index** clause is mandatory if you are writing to an indexed file; e.g.,

```
0810 LET I=0
0820 OPEN (8)"PVX_INDX"
0830 READ (8)IOL=110,ERR=0950
0840 LET I=I+1 ! This reserves an empty record at index 0
0850 CALL "SOMETHING",IOL=0110,ERR=950
0900 WRITE (8,IND=I,ERR=9000)IOL=0110
0910 GOTO 0830
```

The **KEY=num** is mandatory if you are writing to a Keyed file with an external key or to a **DIRECT** or **SORT** file; e.g.,

```
0710 OPEN (7)"PVX_SORT"
0720 READ (6)CUST$,NAME$,*,*,*,*,*,*,*,*,ERR=0750
0730 WRITE (7,KEY=CUST$)
```

No **KEY=** option is allowed if you are writing to a Keyed file whose primary key is composed of data fields embedded in the record data. In Keyed files with multiple keys, the **WRITE** directive will automatically update all alternate keys. For instance, alternate keys 0 [1:1:6] and 1 [2:1:10] are updated as follows:

```
KEYED "PVX_KEYD", [1:1:6],[2:1:10],,256
0210 OPEN (2)"PVX_KEYD"
0220 READ (6)CUST$,NAME$,*,*,*,*,*,START_DT$,CRED_LIM,TERMS,END_DT$,ERR=0250
0230 WRITE (2)IOL=0100
```

ProvideX uses the variables in the variable list either in delimited form or in accordance with any format specified (with headers, etc.). The contents of these fields are used to generate the actual data record. Numeric data converted during a **WRITE** directive does not use the '**DP**' **Decimal Point Symbol** or '**TH**' **Thousands Separator** system parameters for European decimal settings.

The list of variables can refer to an IOList (using **IOL=iolref**) as above. The *iolref* can be the line number or label of the line containing the IOList, or it can be a string containing a compiled IOList. If you omit the list of variables from the **WRITE** directive, ProvideX uses the IOL specified (if any) on your **OPEN** statement for the file.

*Writing to *MEMORY**

A **WRITE** operation will check the last entry in the key table for the key being added before proceeding to the top of the key chain to determine the new entry point. This dramatically increases the speed of writing additional records in sequential order.

You can use **WRITE** and/or **WRITE RECORD** directives to update records in a Memory file using an IOList or a string expression. Records may be inserted by index at a given index number. If the index does not already exist, the record will be automatically appended at the last unused index.

ProvideX will not overwrite existing records. Use the **DOM=** option when you write to a memory file. The following two examples below insert a new record at index 3 without overwriting the current record at index 3. The record that was at index 3 is now at index 4 and the number of records in the file has increased by one.

```
WRITE (14,IND=3)IOL=2010    or
WRITE RECORD (14,IND=3)"DOGCATPIG"
```

To update a given record in a Memory file, use **KEY=** with a given key value:

```
0910 WRITE (14,KEY=KK$,DOM=0920)IOL=2010    or
WRITE RECORD (14,KEY=KK$)A$
```

Format 2: *Write Lock*

WRITE (*chan[,fileopt]*)*varlist*

Use the **WRITE LOCK** format to ensure that once the file has been written to, it remains locked; e.g.,

```
9010 WRITE LOCK (9,ERR=2000)IOL=0110
```

When you use the **LOCK** option, ProvideX doesn't release an extracted record. It maintains the extraction to prevent potential timing problems and maintain counters and totals in batch processing, sparing you the need to re-extract; e.g.,

```
9010 WRITE LOCK (9, KEY=K$)
```



Note: If a serial file is not locked before you write to it, an Error #13: File access mode invalid will occur on the **WRITE** directive. Use **OPEN LOCK** for your serial file to prevent this error from occurring on the **WRITE**.

Example:

```
SERIAL "PVX_SER", ,256
0510 OPEN LOCK (5)"PVX_SER"
0520 READ (6)CUST$,NAME$,ADDR1$,ADDR2$,CITY$,PROV$,POSTAL$,START_DT$,CREDLIM,
0520:TERMS,END_DT$,ERR=0550
0530 WRITE (5)IOL=0090
0540 GOTO 0520
0550 STOP
```

See Also

INSERT Insert New Record in File, *p.162*
UPDATE Update Existing Record in File, *p.351*
WRITE RECORD Write Record, *p.386*
OPEN Open for Processing, *p.232*,
RCD() Function, *p.508*
'XI' System Parameter, *p.696*,
MEMORY Create & Use Memory File, *p.741*

WRITE RECORD Directive

Write Record

Format **WRITE RECORD** (*chan*[,*fileopt*])*contents*\$

Where:

chan Channel or logical file number of the file to which to write.

contents\$ String (literal, expression, or variable) that contains the contents of the record to write.

fileopt Supported file options (see also, [File Options, p.810](#)):
DOM=stmref Missing record transfer
END=stmref END-OF-FILE transfer
ERR=stmref Error transfer
IND=num Record index
KEY=num Record key (see *Automatic Padding* below)
REC=name\$ Record prefix (**REC=VIS(string\$)** can also be used)
RTY=num Number of retries (one second intervals)
SIZ=num Number characters to write
TIM=num Maximum time-out value in integer seconds.

stmref Program line number or label to transfer control to.

Description Use the **WRITE RECORD** directive to add / update a record for a file (logical file number / channel). If the specific record already exists (indexed, direct, or sort files) and you include the **DOM=stmref** option, control transfers to the *stmref*. Otherwise, the specified record is updated.

In Keyed files with multiple keys, the **WRITE RECORD** directive automatically updates all alternate keys. ProvideX supports use of the **WRITE RECORD** directive with ***MEMORY***.

Automatic Padding with KEY=Option

When you use **KEY=string\$:string\$[:string\$][...]** ProvideX automatically pads key segments. This is valid only if you have Keyed files with segmented key definitions. ProvideX right-pads the key segment using \$00\$ (nulls) to the segment's full length. The last segment in a compound key is not padded; e.g.,

Example:

```
KEYED "TEST", [1:1:5]+[2:1:6]+[3:1:8]
READ (1,KEY=A$:B$:C$)
```

is the same as

```
READ (1,KEY=PAD(A$,5,$00$)+PAD(B$,6,$00$)+C$)
```

Writing to *MEMORY*

A **WRITE** operation will check the last entry in the key table for the key being added before proceeding to the top of the key chain to determine the new entry point. This dramatically increases the speed of writing additional records in sequential order.

You can use **WRITE** and/or **WRITE RECORD** directives to update records in a Memory file using an IOList or a string expression. You can add records by index, inserting records at the given index number.

ProvideX will not overwrite existing records. Use the **DOM=** option when you write to a Memory file. The following two examples below insert a new record at index 3 without overwriting the current record at index 3. The record that was at index 3 is now at index 4 and the number of records in the file has increased by one.

```
WRITE (14,IND=3)IOL=2010    or
WRITE RECORD (14,IND=3)"DOGCATPIG"
```

To update a given record in a Memory file, use **KEY=** with a given key value:

```
0910 WRITE (14,KEY=KK$,DOM=0920)IOL=2010    or
WRITE RECORD (14,KEY=KK$)A$
```

Example

```
0010 OPEN (1)"OLDFIL"
0020 OPEN LOCK (2)"NEWFIL"
0030 READ RECORD (1,END=1000)R$
0040 WRITE RECORD (2)R$
0050 GOTO 0030
1000 CLOSE
1010 END
```

See Also

[WRITE Add/Update Data in File, p.383](#)
[INSERT Insert New Record in File, p.162](#)
[UPDATE Update Existing Record in File, p.351](#)



3

System Functions

@()	EVS()	LOG()	RND()
@X() / @Y()	EXP()	LRC()	RNO()
ABS()	FFN()	LST()	SEP()
ACS()	FIB()	MAX()	SGN()
AND()	FID()	MEM()	SIN()
ARG()	FIN()	MID()	SQR()
ASC()	FPT()	MIN()	SRT()
ASN()	GAP()	MNM()	SSZ()
ATH()	GBL()	MOD()	STK()
ATN()	GEP()	MSG()	STP()
BIN()	HSA()	MSK()	STR()
BSZ()	HSH()	MXC() / MXL()	SUB()
CHG()	HTA()	NEW()	SWP()
CHR()	HWN()	NOT()	SYS()
CMP()	I3E()	NUL()	TAN()
COS()	IND()	NUM()	TBL()
CPL()	INT()	OBJ()	TCB()
CRC()	IOL()	OPT()	TMR()
CSE()	IOR()	PAD()	TRX()
CTL()	JUL()	PCK()	TSK()
CVS()	JST()	PFX()	TXH()
DEC()	KEC()	PGM()	TXW()
DIM()	KEF()	POS()	UCP()
DIR()	KEL()	PRC()	UCS()
DLL()	KEN()	PRM()	UPK()
DSK()	KEP()	PTH()	VIN() / VIS()
DTE()	KEY()	PUB()	XEQ()
ENV()	KGN()	RCD()	XFA()
EPT()	LCS()	RDX()	XOR()
ERR()	LEN()	REC()	
EVN()	LNO()	REF()	

Overview

This chapter provides an alphabetically arranged list of all the system functions in ProvideX. Each definition includes the correct syntax (showing associated parameters), values returned, a general description, examples, and sometimes a cross reference to related documentation. The list begins on the following page.

@() Function

Location Function

Formats @(column[,line][,ERR=stmtref])

Where:

column Column number. Integer value indicating a column position on a printer or terminal screen (0 to the number of columns available on the screen -1)

line Optional line number. Integer value indicating a line position on a printer or terminal screen (0 to the number of lines available on the screen -1)

stmtref Program line number or label to transfer control to.

Returns String, position code mapped to a location on an output device.

Description The @() location function is used to print/display text at a specific position on a printer or terminal screen. This function can be used with directives wherever text is to be sent to an output device, most commonly via **INPUT** or **PRINT** statements.

Column 0 represents the column on the far left side of the screen/printer, and line 0 represents the top line. Output is positioned at the column specified on the current line only if the column number is provided. When used with a printer (or print file) and the line number is less than the current line, a new page is started.

Examples The following statement prints the date in the upper left hand corner of the screen with the time starting in column 75 of the top line:

```
PRINT @(0,0), "Date: ", DAY,@(75),TIM
```

This prompts for information on the left side, 5 lines from the top:

```
INPUT @(0,5), "Enter favorite sport:",@(30),SPORT$,@(0,6), "Thanks"
```

This prints the date right-justified in the upper right hand corner of the screen:

```
LET D$=DTE(0:"%Dl %Ms %D")
PRINT 'BLUE',@(80-LEN(D$),0),D$,
```

@X() / @Y() Functions Convert X/Y Coordinates

Formats 1. Return Column Position: @X(col[,chan][,ERR=stmtref])

2. Return Line Position: @Y(line[,chan][,ERR=stmtref])

Where:

chan Channel or logical file number of the device or file.

col,line Standard print positions. Numeric expressions. Column & line numbers.

stmtref Program line number or label to transfer control to.

Returns Integers (x-axis coordinate for column and y-axis coordinate for line, in graphical units – similar to pixels).

Description Use these functions to convert column and line number addresses for graphics output into x- and y-axis coordinates in graphical units (similar to pixels). The values returned are integers. In the example below, the functions return 16 graphical units as the coordinate for column 1 and 60 for line 2:

```
->0010 LET A=1, B=2
-:0020 PRINT @X(A),@Y(B)
-:RUN
16 60
```

These functions take the column/line numbers and, based on your given channel, return the corresponding graphics coordinates. Numeric variables, fractions and negative numbers are allowed as column and line values.

If you do not include a channel, the default is 0 *zero*, the terminal/console. If you use either of these functions in a print statement, the output will go to the channel or logical file number you specify in the **PRINT** directive; i.e., to a terminal, file or printer.

Examples

```
1010 PRINT @(5,5), "CUSTOMER LISTING",
1020 PRINT 'FILL'(0,0), 'RECTANGLE' (@X(4),@Y(5),@X(22),@Y(6)),
```

When the example above is **RUN**, the words "CUSTOMER LISTING" are displayed on the screen, outlined by a rectangle. The @X(*col*) values start the rectangle at the graphical equivalent of column 4 (one column before the text) and end it one column after the text. The @Y(*line*) values are equivalent to lines 5 and 6, and they define the rectangle as 1 line high starting at the top of line 5.

ABS() Function

Absolute Value

Format **ABS(num[,ERR=stmtref])**

Where:

num Numeric expression whose absolute value is to be returned.

stmtref Program line number or label to transfer control to.

Returns Numeric, absolute value of expression. Always positive or zero.

Description The **ABS()** function returns the absolute value of the numeric expression *num*. The value returned is a positive numeric value or zero. For instance, the absolute value of the numeric expression (**X**) is returned as follows:

- **ABS (X)** ... If **X**>0 ... Returns **X**
- **ABS (X)** ... If **X**<0 ... Returns **X** * -1 (positive result)
- **ABS (X)** ... If **X**=0 ... Returns **X** (zero).

Examples

```
0010 INPUT "Give me your first number ",X
0020 INPUT "Give me another one ",Y
0030 PRINT "The difference is ",ABS(X-Y)
-:RUN
Give me your first number 12.345
Give me another one 23.456
The difference is 11.111
```


ACS() Function

Return Arc-Cosine

Format `ACS(num[,ERR=stmtref])`

Where:

num Numeric expression (range: -1 to +1) whose Arc-Cosine is to be returned.

stmtref Program line number or label to transfer control to.

Returns Numeric value between 0 *zero* and π (pi=3.14159...)

Description The **ACS()** function returns the Arc-Cosine of the numeric expression *num*. It returns a value between 0 *zero* and π (pi) rounded to the current **PRECISION** in effect. This is the inverse of the **COS()** function.

The value of your numeric must be in the range of -1 to +1 inclusive. Otherwise ProvideX returns

Error #40: Divide check or numeric overflow.

Examples

```
0060 FOR I=-1 TO 1 STEP .1
0070 PRINT ACS(I),
0080 NEXT
0090 PRINT " DONE"
0100 END
-:run
3.14 2.69 2.5 2.35 2.21 2.09 1.98 1.88 1.77 1.67 1.57 1.47 1.37 1.27 1.16
1.05 0.93 0.8 0.64 0.45 0 DONE
```

AND() Function

Logical AND

Format **AND**(value1[\$],value2[\$][,ERR=stmtref])

Where:

stmtref Program line number or label to transfer control to.

value1[\$] Compared values. String or numeric expressions/variables. If strings,
value2[\$] *value1\$* must be the same length as *value2\$*

Returns Result of bit-wise logical 'AND' comparison of two expressions/variables.

Description The **AND()** function performs a bit-wise 'AND' comparison of two string or numeric expressions/variables, and generates a value as a result. The length of the two string expressions must be equal or ProvideX returns an Error #46: Length of string invalid.

<i>Binary</i>	<i>Result</i>
0 and 0	0
1 and 0	0
0 and 1	0
1 and 1	1

See Also [IOR\(\) OR Comparison, p.460](#)
[XOR\(\) Exclusive OR Comparison, p.554](#)

Example

```
0040 READ (1,END=1000) F$
0050 R$=AND(F$(1,2),$7F7F$) ! Turn off high bit
0060 ....
```

```
BITSS=$03$
IF AND(BITSS,$02$)=$02$ THEN PRINT "bit 2 is on"
```

ARG() Function

Command-Line Argument

Format **ARG(position[,ERR=stmtref])**

Where:

position Position of the argument you want returned. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns String, value of start up argument.

Description The **ARG()** function returns the string value of one of the arguments specified in the operating system command that launched ProvideX. The numeric expression *position* indicates which argument is to be returned:

-4 Original command line used to start ProvideX.

-3 ProvideX Activation Key directory name.

-2 ProvideX Library Directory name.

-1 INI file in use (fully resolved path to the INI file). Returns "" on UNIX/Linux.

0 The command that launched ProvideX

1, 2, 3, etc. Other arguments after the command

If you refer to an invalid argument, ProvideX returns:

Error #41: Invalid integer encountered (range error or non-integer)

See Also [NAR System Variable, p.567](#),
ProvideX Installation Guide, Launching ProvideX

Example *Given*

```
PVX -SZ=20 -ARG TOM JONES
```

Then

ARG(-1) *yields* the INI filename in use for the current session

ARG(0) *yields* PVX (the command that launched ProvideX)

ARG(1) *yields* TOM

ARG(2) *yields* JONES

ARG(3) *yields* Error #41: Invalid integer encountered...

ASC() Function

Get Internal Character Value

Format `ASC(char$[,ERR=stmtref])`

Where:

char\$ Single-character whose ASCII value is to be returned. String expression.

stmtref Program line number or label to transfer control to.

Returns Integer, ASCII character set number, range 0 to 255.

Description The **ASC()** function returns the internal numeric value of a given character *char\$*, based on its position in the ASCII character set/table. The value returned is an integer, range 0 to 255 (ASCII character set number). Only the first character in the string is converted to its internal value.

Examples

```
?ASC("A")! yields 65
```

```
string$="a";?ASC(string$) ! yields 97
```

```
?ASC("abc") ! also yields 97 (the ASCII value for "a")
```

ASN() Function

Returns Arc-Sine Function

Format **ASN(num[,ERR=stmtref])**

Where:

num Numeric expression (range: -1 to +1) whose ARC-Sine is to be returned.

stmtref Program line number or label to transfer control to.

Returns Numeric value, range $-\pi/2$ to $+\pi/2$.

Description The **ASN()** function returns the Arc-Sine of the given numeric expression *num*. The function returns a value between $-\pi$ (negative pi) divided by 2 and $+\pi$ (+pi) divided by 2, rounded to the current **PRECISION** in effect. This is the inverse of the **SIN()** function.

If *num* is not in the range -1 to +1 inclusive, the result will be:

Error #40: Divide check or numeric overflow

Examples

```

0060 FOR I=-1 TO 1 STEP .1
0070 PRINT ASN(I),
0080 NEXT
0090 PRINT "DONE"
0100 END
-:run
-1.57-1.12-0.93-0.78-0.64-0.52-0.41-0.3-0.2-0.1 0 0.1 0.2 0.3 0.41 0.52
0.64 0.7 8 0.93 1.12 1.57 DONE

```

ATH() Function

Convert Hex

Format `ATH(hex_string$[,ERR=stmtref])`

Where:

hex_string\$ Hex string expression to be converted to ASCII. The string must be an even number of bytes in length and consist of only the characters 0 through 9 and A through F.

stmtref Program line number or label to transfer control to.

Returns ASCII value corresponding to string of hex data.

Description The **ATH()** function converts string containing valid hexadecimal data to ASCII. The ASCII string returned by the **ATH()** function is the converted value of each set of two hex characters to one ASCII character.

Examples

```
-:h$="414243"
```

```
-:?ath(h$)
```

```
ABC
```

```
ATH ("414240") ! yields AB@ (Hex 40= "@")
```

ATN() Function

Return Arc-Tangent

Format **ATN(num[,ERR=stmtref])**

Where:

num Numeric expression whose Arc-Tangent is to be returned.

stmtref Program line number or label to transfer control to.

Returns Numeric value, range $-\pi/2$ to $+\pi/2$.

Description The **ATN()** function returns the Arc-Tangent of the numeric expression *num*. It will return a value between $-\pi$ (negative pi) divided by 2 and π divided by 2, rounded to the current **PRECISION** in effect. This is the inverse of the **TAN()** function.

Examples

```

0060 FOR I=-10 TO 10 STEP 1
0070 PRINT ATN(I),
0080 NEXT
0090 PRINT " DONE"
0100 END
-:RUN
-1.47-1.46-1.45-1.43-1.41-1.37-1.33-1.25-1.11-0.79 0 0.79 1.11 1.25 1.33
      1.37 1.41 1.43 1.45 1.46 1.47 DONE

```

BIN() Function Binary String from Numeric Value

Format **BIN(num,len[,ERR=stmtref])**

Where:

len Length of the string to be returned. Use a numeric expression. Integer value.

num Numeric value to convert to a string. Use a numeric expression. Integer value.

stmtref Program line number or label to transfer control to.

Returns Binary value of ASCII string corresponding to the numeric expression.

Description The **BIN()** function converts the numeric expression *num* to an ASCII string (as its binary value). The string returned will be the length specified, *len*. The value is right justified in the resultant string.

The value is converted to two's complement format before the string is generated. If the number was negative the leading bits will all be set to **ON** (binary 1).

2 is binary 00000010

1 is binary 00000001

0 is binary 00000000

-1 is binary 11111111

-2 is binary 11111110

Examples

```
B$=BIN(40,1);PRINT HTA(B$) ! yields $28$ 00101000
```

```
B$=BIN(40,2);PRINT HTA(B$) ! yields $0028$ 00000000 00101000
```

```
B$=BIN(2048,2);PRINT HTA(B$) ! yields $0800$ 00001000 00000000
```

```
B$=BIN(-64,2);PRINT HTA(B$) ! yields $FFC0$ 11111111 11100000
```


BSZ() Function

Bank Memory Size

Format **BSZ(num[,ERR=stmtref])**

Where:

num Bank number (*ignored*). Numeric expression.

stmtref Program line number or label to transfer control to.

Returns 0 *zero* in most ProvideX implementations.



Note: This function is included for compatibility with other languages.

Description The **BSZ()** function is primarily included in ProvideX for compatibility with other Business Basic languages. The value it returns is 0 *zero* in most ProvideX implementations.

CHG() Function Notify if Variable Has Changed

- Formats
1. *Notify & Change Status*: **CHG**(varlist[,ERR=stmtref])
 2. *Non-Destructive Read*: **CHG(READ** varlist[,ERR=stmtref])

Where:

stmtref Program line number or label to transfer control to.

varlist Comma-separated list of variable names to be tested.

Returns String, listing variables that have changed.

Description Use the **CHG()** function to determine if the contents of a variable has changed. Re-assignment of the same data is still considered a change. Given a comma-separated list of variable names, the **CHG()** function will return a comma-separated list of those variables that have changed. Use += (add to), -= (subtract from), and other operators with the **CHG()** function.



Note: You should not use the **CHG()** function with NOMADS applications (except for variables not referenced by the panel), because when you set the REFRESH_FLG, NOMADS itself uses the **CHG()** function to determine which variables are to be refreshed.

Format 1: *Notify & Change Status*

CHG(varlist[,ERR=stmtref])

The "changed" status of a variable is affected when the data in the variable changes or when the **CHG()** function is applied. When this format of the **CHG()** function is used, it will reset the "changed" status of the variables listed; e.g.,

```
0010 PRINT CHG( "A,B,D$,E$" )
0020 LET A=1.234
0030 LET E$="MIKEY"
0040 PRINT CHG( "A,B,D$,E$" ) ! Prints A,E$
0050 PRINT CHG( "A,B,D$,E$" ) ! Prints nothing
```

Format 2: *Non-Destructive Read*

CHG(READ varlist[,ERR=stmtref])

The **CHG(READ)** format allows you to do a non-destructive read of the value returned by the **CHG()** function. The "changed" status will not be reset when the function is applied; e.g.,

```
0010 PRINT CHG( "A,B,D$,E$" )
0020 LET A=1.234
0030 LET E$="MIKEY"
0040 PRINT CHG(READ "A,B,D$,E$" ) ! Prints A,E$
0050 PRINT CHG( "A,B,D$,E$" ) ! Prints A,E$
```

CHR() Function

ASCII Character of Value

Format **CHR(num[,ERR=stmtref])**

Where:

num Value of the character to return. Numeric expression. Range: integer from 0 to 255 (the character's number in the ASCII table).

stmtref Program line number or label to transfer control to.

Returns Single ASCII character.

Description The **CHR()** function returns a single-character ASCII string of the numeric expression defined by *num*. The value of *num* must be an integer in a range from 0 to 255 (the position of the character in the ASCII table). The first printable character in the ASCII character set is the blank, \$20\$, which is **CHR(32)**.

See Also [ASC\(\) Get Internal Character Value, p.396](#)

Examples
LET X\$=CHR(65) ! (Sets X\$ to "A")
LET X\$=CHR(33) ! (Sets X\$ to "!")

CMP() Function

Compress Data

Format **CMP(string[,ERR=stmtref])**

Where:

string Original data to be compressed.

stmtref Program line number or label to transfer control to.

Returns Compressed data string.

Description The **CMP()** function uses standard ZLib compression libraries to compress a data string. To expand (uncompress) the data, use the **UCP() Function, p.547**. **TCB(195)** will return 1 if ZLib support is available.

The data returned from the **CMP()** function includes a single header byte (value between \$01\$ and \$FF\$) to facilitate ZLib uncompression routines. This represents a multiplier value that can be used against the length of the compressed data to estimate uncompressed data size (basically, *original size/compressed size* rounded up and capped at 255). When expanding the compressed data using the **UCP()** function, the length of the compressed data will be multiplied by this header byte to determine an estimated uncompressed buffer size.



Note: Since the **CMP()** and **UCP()** compression routines are not supported on all platforms, systems using these functions may not be fully portable.

Typically, the compressed data will be smaller than the original data; however, in rare instances or when dealing with short strings, the compressed data might be longer. Also be aware that data returned from the compression logic may contain any potential character; therefore, if writing to a data file, do not use field delimiters since they may occur within the compressed data.

Interfacing with Other ZLib-compliant Utilities

The header byte should be removed from the compressed data if it is to be used outside of ProvideX.

Example

```
0100 ! ^100 - CMP and UCP functions
0110 Original$=dim(512,"String to compress ")
0120 Compressed$=cmp(Original$)
0130 print "Length of Original:  ", 'BR', str(len(Original$)), 'ER'
0140 print "Original String:      ", 'BR', Original$, 'ER'
0150 print "Length of Compressed:", 'BR', str(len(Compressed$)), 'ER'
0160 print "Compressed String:   ", 'BR', cvs(Compressed$,16:"~"), 'ER'
0170 print
0180 !
0200 ! ^100 - Due to Zlib overhead, not all strings yield smaller strings
```

```
0210 for StrLen=16 to 64 step 8
0220 Original$=dim(StrLen,"This is a short string")
0230 Compressed$=cmp(Original$)
0240 o=len(Original$),c=len(Compressed$)
0250 print "Orig:",o," CMP:",c," ",tbl(o>c,'red'+ "Not Shorter"+'RM',"")
0260 next
```

See Also [PCK\(\) Pack Numeric Data, p.498](#)
[UCP\(\) UnCompress Data, p.547](#)

COS() Function

Return Cosine

Format `COS(num[,ERR=stmtref])`

Where:

num Numeric expression whose Cosine is to be returned.

stmtref Program line number or label to transfer control to.

Returns Value between -1 and +1.

Description The **COS()** function returns the Cosine of the numeric expression. It returns a value between -1 and +1, rounded to the current **PRECISION** in effect. **ACS()** is its inverse function.

Examples

```
00010 PRINT 'CS'  
00020 X=0,Y=80  
00030 LASTX=0, LASTY=80  
00040 FOR I=-20 TO 1 STEP .5  
00050 X=X+25  
00060 Y=50*COS(I)+80  
00070 PRINT 'LINE' (LASTX, LASTY, X, Y)  
00080 LASTX=X  
00090 LASTY=Y  
00100 NEXT
```

CPL() Function

Compile String

Format **CPL(statement\$[,ERR=stmtref])**

Where:

statement\$ String containing the statement to compile.

stmtref Program line number or label to transfer control to.

Returns ProvideX compiled format of the statement.

Description The **CPL()** function returns the ProvideX internal (compiled) format of the statement provided in the string argument *statement\$*.

Example

```
MYSRS_IOL$=CPL( "IOLIST A$,B$,C$" )
READ DATA FROM CUR_RECORD$ TO IOL=MYSRS_IOL$
```

CRC() Function

Cyclic-Redundancy-Check

Format `CRC(chars[,basis$][,ERR=stmtref])`

Where:

basis\$ Optional *initial* value to be used as the basis of the CRC. It must be *two* characters long if included in the statement. Default: \$0000\$.

chars\$ String of characters on which to calculate a cyclic redundancy check.

stmtref Program line number or label to transfer control to.



Note: The **CRC()** function is used primarily for generating transmission checksums on synchronous communications.

Returns Cyclic redundancy checksum (in internal format).

Description The **CRC()** function returns the cyclic redundancy checksum of a string of characters.

Use the initial value (*basis\$*) to generate an overall **CRC()** of multiple strings. See the example below, where `CRC(B$)` with an initial value of `CRC(A$)` will be the same as `CRC(A$+B$)`. If you omit the initial value, ProvideX uses the default value, \$0000\$.

Examples

```
A$="Hello",B$="World"
0010 LET C$=CRC(A$)
0020 LET C$=CRC(B$,C$)
```

Yields the same result as:

```
0030 LET C$=CRC(A$+B$)
```

That is:

HTA (C\$)	Value
After line 0010	\$F353\$
After line 0020	\$6053\$
After line 0030	\$6053\$

CSE() Function

Case Compare

Format `CSE(expression[$],compare1[$],compare2[$] ...[,ERR=stmtref])`

Where:

`compare1[$]` Comma-separated list of numeric or string expressions for
`compare2[$]` ... comparison with `expression[$]`.

`expression[$]` Numeric or string expression to be used for comparison.

`stmtref` Program line number or label to transfer control to.

Returns Integer representing the sequence number of a match, or 0 *zero* if no match.

Description The **CSE()** function is used to compare an expression with a series of values. If the expression has a match, the resulting number will point to the location of the matched value in the series; i.e., if *expression* matches the 5th value in the *compare* series, then **CSE()** returns 5. If no match is found, **CSE()** returns 0.

Examples

```
1000 INPUT X$
1010 ON CSE(CTL,1,2,1000,1020,1030) GOTO NOFKY,FK1,FK2,QRV_CST,DEL_CST,ADD_CST
```

In the example above, if the value of CTL is 1000 then the **CSE()** function returns 3; if the value of CTL is 1030, then 5 is returned; etc.

The example below shows strings in the comparison.

```
1010 ON CSE(TEST$, "ANTIDISESTABLISHMENTARIANISM", "DOG", "CAT", "PIG") GOTO 1000,
1010:2000,3000,4000
```

See Also [TBL\(\) Convert String Via Table, p.532](#)
[SWITCH..CASE Directive, p.331](#)
[CASE Directive, p.42](#)

CTL() Function

Return CTL Definition

Format

1. Return CTL Definition: **CTL(chan,input[\$][,ERR=stmtref])**
2. Return Input Sequence: **CTL(READ chan,ctlval[,ERR=stmtref])**

Where:

chan Channel or logical file number. Usually 0 zero, the terminal.

ctlval CTL value whose input sequence is to be returned.

input[\$] Input sequence. Either the keyboard character string received or the invalid CTL value. String or numeric expression.

stmtref Program line number or label to transfer control to.

Returns

CTL code associated with input, or input sequence associated with **CTL**.

Description

The **CTL()** function is designed for use in programs that interact directly with the terminal. It provides access to the **CTL** lookup tables maintained internally in ProvideX. When an input sequence is passed to **CTL()**, it returns the **CTL** values associated with the input; e.g., `PRINT CTL(0,$000002e$)` returns -1007. When a **CTL** value is passed to **CTL(READ ..)**, it returns the input sequence associated with the control.

See Also

[DEFCTL Directive, p.78](#)
[Negative CTL Definitions, p.817](#)

Examples

The following example reads an uppercase string into Y\$ without editing and stops on a **CTL** code > 0:

```
0020 OPEN (1) FID(0); Y$=""
0030 READ RECORD (1,SIZ=1)X$
0040 IF X$<" " OR X$=$7F$ THEN GOTO 0100
0050 LET X$=UCS(X$),Y$=Y$+X$
0060 PRINT X$; GOTO 0030
0100 LET C=CTL(0,X$,ERR=0050); GOTO 0130
0110 READ RECORD (1,SIZ=1)X1$
0120 LET X$=X$+X1$; GOTO 0100
0130 IF C>0 THEN GOTO 1000 ! Found it
0140 LET C=CTL(0,C,ERR=0160) ! Check alternates
0150 GOTO 0130
0160 PRINT 'RB'; GOTO 0030
1000 PRINT "DONE"; STOP
```

In the example below, the **CTL()** function returns a string value containing a list of control sequences for a specified CTL value. The string is made up of a series of fields where the 1st byte preceding each field indicates the number of characters in the control sequence; the rest of the field shows the actual control sequence. To print control sequences for function keys **F1** through **F4**:

```
0010 FOR CTL_CODE=1 TO 4
0020 PRINT 'LF',STR(CTL_CODE:"#0")," ",
0030 LET X$=CTL(READ 0,CTL_CODE)
0040 IF X$="" THEN GOTO 0070
0050 LET X=DEC(X$(1,1)); PRINT PAD(HTA(X$(2,X)),10),
0060 LET X$=X$(X+2); GOTO 0040
0070 NEXT
0080 PRINT " DONE"; STOP
-:RUN
1 000070
2 000071
3 000072
4 1B          000073      00FF82      DONE
```

If the expression is numeric, ProvideX returns any alternate CTL value that has been assigned; e.g., `PRINT CTL(0,000070)` returns 1 (CTL=1 or the **F1** key).

CVS() Function

Convert String

Formats

1. *Reformat String*: **CVS(data\$, [cvs_code[:ctl_char\$], ...], [ERR=stmtref])**

2. *Return Accent Table*: **CVS(*)**

Where:

* *Asterisk*. Returns the current 256-byte accent translation table.

ctl_char\$ Optional control character. String expression. Default: blank character

cvs_code Conversion type. Numeric expression. See the chart below.

data\$ Data to convert. String expression.

stmtref Program line number or label to transfer control to.

Returns

Converted value of string expression, or accent translation table.

Description

The **CVS()** function converts a string of data to different forms based on the numeric values used for *cvs_code* (e.g., to convert to upper case, stripped string, etc). The numeric expression tells ProvideX the type of conversion to perform. If you append a colon ":" and string value, the control character used in the conversion will be the first character of the string. Otherwise, it will be a space.

You can also use the function to return the current accent translation table and do translations based on it. For more information, refer to the **DEF systab= Directives, p.74**.

The value of *cvs_code* is made up of a series of binary values indicating:

Value	Conversion to Take Place:
1	Strip leading control characters
2	Strip trailing control characters
4	Convert string to upper case
8	Convert string to lower case
16	Replace non-printable characters with the control character
32	Replace multiple occurrences of the character with one
64	Replace \$ with a defined currency symbol, comma with a defined thousands separator, and period with a defined decimal point. See also, 'CU' = System Parameter, p.660, 'DP' = System Parameter, p.662, 'TH' = System Parameter, p.690
128	Replace a defined currency symbol, thousands separator and decimal point with \$, comma, and period respectively (inverse of value 64 above).

Value	Conversion to Take Place:
256	Convert string to mixed case (first letter of each word is upper case, the rest of the letters are lower case).
512	Translate the string expression provided in <i>data</i> \$ based on the current accent translation table.

Examples

```
PRINT "|",CVS(" The Cat ",1:" "),"|"
```

```
yields |The Cat |
```

```
PRINT "|",CVS(" The Cat ",2:" "),"|"
```

```
yields |The Cat|
```

```
PRINT "|",CVS(" The Cat ",39),"|"
```

```
yields |THE CAT| Note: 32+4+2+1=39 (you can use a sum)
```

```
PRINT CVS("$$$123",32:"$")
```

```
yields $123
```

See Also

STP() Strip Leading/Trailing Characters, p.523,

LCS() Return Lowercase String, p.472,

UCS() Return Upper Case String, p.546

DEF systab= Directives, p.74.

DEC() Function

Get Binary of String

Format `DEC(string$[,ERR=stmtref])`

Where:

stmtref Program line number or label to transfer control to.

string\$ Character string or variable containing value to be converted to binary.

Returns Two's complement binary equivalent of the string.

Description The **DEC()** function converts a string to its corresponding binary equivalent. (The value returned is a two's complement binary integer which corresponds to the value of the string.)

Examples

```

0020 LET A$="a"; PRINT DEC(A$),
0030 LET B$=$0040$; PRINT " | ",DEC(B$),
0040 LET C=DEC("A"); PRINT " | ",C,
0050 LET A=DEC($FE$); PRINT " | ",A,
0060 PRINT " | DONE"
-:RUN
97 | 64 | 65 | -2 | DONE

```

DIM() Function Generate String/Get Array Size

- Formats
1. *Generate or Initialize String*: `DIM(len[,fill$][,ERR=stmtref])`
 2. *Read Total elements in Array*: `DIM(READ NUM(array_name[$][,subscript]))`
 3. *Read Max. in Array*: `DIM(READ MAX(array_name[$][,subscript]))`
 4. *Read Min. in Array*: `DIM(READ MIN(array_name[$][,subscript]))`

Where:

array_name[\$] Name of a previously dimensioned array.

fill\$ Text string or value which will be used to fill the variable up to the length specified. String expression.

len Desired length of the string variable. Numeric expression, integer.

stmtref Program line number or label to transfer control to.

subscript Array's dimensions (first, second, or third).

Returns Initialized string, or information about dimensions of array.

Description Use the **DIM()** function to generate or initialize a string. You can also use it to obtain information about the size of an array you have already defined using the **DIM** directive.

Format 1: Generate or Initialize String

`DIM(len[,fill$][,ERR=stmtref])`

Use this format to generate or initialize a string whose length you specify. The function fills the string by using the value for *fill\$*.



Note: This format of the function behaves almost like the **DIM** directive's format to initialize strings. However, instead of repeating only the first character of the string, as in the directive, the function repeats the complete value of the fill string. (Refer to the [DIM Directive, p.86](#), for the format to initialize strings).

If you omit the fill string, ProvideX uses spaces as fill characters; e.g.,

```
0100 PRINT " : ",DIM(11,"*Cat" ) , " : "
->RUN
*Cat*Cat*Ca:
```

```

0100 PRINT 'CS',"A Title",@(0,1),DIM(80,"-")
0110 REM The DIM( ) above returns a string of hyphens "-" 80 characters long
->RUN
A Title
----- ... etc.

```

See Also ['+S' & '-S' Mnemonics, p.637.](#)

Formats 2,3,4: Read Dimensions (Size) of Array

Use Formats 2, 3, and 4 of the **DIM()** function (listed below) to read a given array's total number of elements, the minimum element's number and the maximum element's number. If the variable you specify is not an array, **NUM** and **MIN** will return 0, **MAX** will return -1.

DIM(READ NUM(array_name[\$][,subscript]))

Use the **DIM(READ NUM())** function to read the total number of elements in a dimensioned array; e.g.,

```

0100 DIM X[0:15]
0110 PRINT DIM(READ NUM(X))
->run
    16

```

DIM(READ MAX(array_name[\$][,subscript]))

Use the **DIM(READ MAX())** function to read the maximum element number in a dimensioned array; e.g.,

```

0110 DIM X[0:15]
0120 PRINT DIM(READ MAX(X))
->run
    15

```

DIM(READ MIN(array_name[\$][,subscript]))

Use the **DIM(READ MIN())** function to read the minimum element number in a dimensioned array; e.g.,

```

0110 DIM X[0:15]
0120 PRINT DIM(READ MIN(X))
->run
    0

```


DIR() Function

Get Current Directory

Format `DIR(disk_id[$][,ERR=stmtref])`

Where:

`disk_id[$]` Disk to check. String or numeric expression.

`stmtref` Program line number or label to transfer control to.

Returns String, name of current directory.



Note: This function is primarily for use in PVX for Windows. Under UNIX, ProvideX assumes that there is only one disk drive and will only returns the current directory.

Description The **DIR()** function returns a string naming the current directory for the disk drive specified. Use either a string or numeric value to specify the drive.

If a *string* is given, the first character indicates the drive (A, B, C and so on). A null ("") string returns the name of the current directory for the current drive. If a *numeric* value is given, it must be an integer in a positive range starting at zero. A value of zero (0) indicates the first drive, 1 the second ...

When reporting items based on the *universal naming convention*, this function returns the UNC-style pathname; i.e., for a current directory of `\\server\share\path\ DIR()` returns `\path\`.

Examples The following illustrates use of the **DIR()** function in a Windows environment.

```

0100 PRINT "Available Drives by Drive Number"
0110 FOR X=0 TO 25 ! Check for Drives 0 to 25
0120 LET A$=DIR(X,ERR=*NEXT); PRINT STR(X)," ",A$
0130 NEXT X
0200 PRINT "Available Drives by Drive Letter"
0210 FOR X=65 TO 90 ! Check for Drives A to Z
0220 LET A$=DIR(CHR(X),ERR=*NEXT); PRINT CHR(X)," ",A$
0230 NEXT X
->run
Available Drives by Drive Number
0  \
2  \Program Files\Sage Software\ProvideX\
3  \WINDOWS\
6  \applicationlogos\
7  \
Available Drives by Drive Letter
A  \
C  \Program Files\Sage Software\ProvideX\
D  \WINDOWS\
G  \applicationlogos\
H  \

```

DLL() Function

Call Windows DLL

Formats

1. *Load Library*: `lib_num=DLL(ADDR lib_string$[,ERR=stmtref])`
2. *Unload Library*: `DLL(DROP lib_num[,ERR=stmtref])`
3. *Get Function Address*: `fnc_addr=DLL(FIND lib_num,fnc_name$[,ERR=stmtref])`
4. *Call Using String*: `DLL(lib_string$,fnc_name$,arg[,arg,arg...][,ERR=stmtref])`
5. *Call by Library Number*: `DLL(lib_num,fnc_name$,arg[,arg,arg...][,ERR=stmtref])`
6. *Call Using Function Address*: `DLL(*,fnc_addr,arg[,arg,arg...][,ERR=stmtref])`

Where:

- arg, arg...* Parameters or arguments to pass to the function. The number and type of arguments you use vary from function to function. See [DLL\(\) Parameters, p.420](#).
- fnc_addr* Address of the desired function. Numeric expression.
- fnc_name* Name of the function. The function name is the *case-sensitive* entry point in the DLL. Some API functions have an appended letter A for ASCII, others have an appended w for wide character UNICODE. String expression.
- lib_num* Internal library identifier used to reference a *loaded* library. Numeric expression.
- lib_string\$* Name of the .dll or .exe that contains the function to be used. String expression.
- stmtref* Program line number or label to transfer control to.

Returns

Windows DLL (Dynamic Link Library), UNIX/Linux shared object module, or address to a function within an external library.

Description

Use the ProvideX **DLL()** function to access functions within DLLs that are external to ProvideX. This function is available under MS Windows as well as most UNIX/Linux environments (in which case, **TCB(196)** returns 1 if the function is supported). It's similar to a call to execute external functions from inside ProvideX applications. The ProvideX **DLL()** function will also return any function identifiers and addresses, as defined by the DLL routine.

DLLs do not need to be registered in Windows to be used in ProvideX. Under WindX, it is important to note that *DLLs run on the host*. Use a call to a WindX program to invoke a DLL.

Use the function **DLX()** to return 32-bit values from within a 16-bit Windows environment. For further information on calling DLLs and other external components from ProvideX, refer to the *User's Guide, Chapter 9*.

See Also [User's Guide, Calling DLLs from ProvideX, p.214](#)
[MEM\(\) Return Memory Value, p.479](#)

Format 1: *Load Library*

```
:lib_num=DLL(ADDR lib_string$[,ERR=stmtref])
```

Use the **DLL(ADDR ...)** format to address or load the DLL. This lets you load and lock a library into memory and obtain the addresses of its functions and the internal identifier. You can then use this identifier on subsequent **DLL()** functions to avoid having to reload the DLL. Note that the return value is the handle to the DLL and should be used in all subsequent **DLL()** calls.



Note: In some instances (e.g., when the DLL maintains internal data structures) it is mandatory to keep the DLL loaded in order to use or call the function. When in doubt, load the DLL (not needed for Windows API DLLs). Keeping the DLL loaded has memory consequences, but commonly lets you gain access speed.

Format 2: *Unload Library*

```
DLL(DROP lib_num[,ERR=stmtref])
```

Use the **DLL(DROP ...)** format to unload a loaded DLL when you no longer need it. Note that if you **LOAD** a DLL, you must **DROP** or unload it to free up the memory that was allocated to it.

Format 3: *Get Function Address*

```
fnc_address=DLL(FIND lib_num,fnc_name$[,ERR=stmtref])
```

You can obtain the address of a **DLL()** function in an addressed (loaded) library by using the **FIND** format. This format asks the **DLL()** function to return its current address, which you can then use in subsequent calls. The address is only valid while the library is loaded.

In some cases, you may need to use the return value as a memory pointer. The **MEM()** function can be used to obtain memory information (address, contents, etc.).

When you call an external function and pass arguments to it, you can identify it using a string, a library number, or a function address. For more information, refer to [DLL\(\) Parameters, p.420](#) and [MEM\(\) Return Memory Value, p.479](#).

Format 4: *Call Using String*

```
DLL(lib_string$,fnc_name$,arg[,arg,arg..],[,ERR=stmtref])
```

Use this format to call the **DLL()** function using strings to identify the library and function by name.

Format 5: Call by Library Number

DLL(lib_num, fnc_name\$, arg[, arg, arg...][, ERR=stmtref])

Use this format to call the **DLL()** function using a numeric to identify the library by its internal identifier and the function name.

Format 6: CALLs Using Function Address

DLL(*, fnc_address, arg[, arg, arg...][, ERR=stmtref])

Use this format to call a **DLL()** function in a loaded library using the internal function address as a memory pointer.



Note: The address is only valid while the library is loaded.

DLL() Parameters

DLLs normally expect one of two types of parameters: integers and pointers. The arguments/parameters you use when you call the **DLL()** function are passed to the function in the following ways:

Example	Type	32-Bit Data Format Passed	16-Bit Data Format Passed
X\$	Strings	Address of string	Address of string
X	Numeric Variables	Double word value (32-bit)	Double word value (32bit)
X%	Integer Variables	16-bit value passed as 32-bit	Single word value (16bit)
INT(X+1)	INT() Function	Standard 32 bit	Single word value (16 bit)
X+Y	Numeric Expression	32-bit value	32-bit value

DSK() Function

Get Current Disk Drive

Format `DSK(disk_id[$][,*][,ERR=stmtref])`

Where:

* *Asterisk*. Returns the name of the volume (rather than the drive).

`disk_id[$]` Disk to check. String or numeric expression.

`stmtref` Program line number or label to transfer control to.

Returns Current disk drive identifier



Note: This function is primarily for use in PVX for Windows. Under UNIX, ProvideX assumes that there is only one disk drive and returns a null ("") string.

Description

The **DSK()** function checks for the existence of a disk drive and returns the current disk drive identifier, volume name. Use either a string or numeric value to specify the drive. If a string is given, the first character indicates the drive (A, B, C, etc). A null ("") string returns the current disk identifier.

This function returns the operating system's drive prefix ("A: ", "B: ", etc.). If you place an *asterisk* * after `disk_id`, the function returns the name of the volume mounted on your drive, if any; e.g., `DSK("C",*)` or `DSK(2,*)`. If the disk drive or volume doesn't exist, ProvideX returns an `Error #17: Invalid file type or contents`.

If a numeric value is given, then the value must be a positive integer in a range starting at zero. Use 0 *zero* as the value for the first drive, 1 for the second, and so on.

When reporting items based on the *universal naming convention*, this function returns the UNC-style pathname; i.e., for a current directory of `\\server\share\path` **DSK()** returns `\\server\share\`.

Examples

```
->?dsk(" ")
C:
->?dsk(0)
A:
```

DTE() Function

Convert Date

Formats

1. Convert Numeric Date: **DTE(jul_date[,time][:fmt\$][,ERR=stmtref])**

2. Convert Date String: **DTE(date\$[:fmt\$][,ERR=stmtref])**

Where:

date\$ Date string in the same format as the **DAY_FORMAT** (MM/DD/YY).

fmt\$ Format of the date to be returned. If omitted, the default format is %Mz/%Dz/%Yz (date formatted as MM/DD/YY).

jul_date Julian date to convert. Numeric expression. If the value is zero, the current date is used. DTE(-1) returns a null ("") string if 'BY'=0.

stmtref Program line number or label to transfer control to.

time Time value containing hours and fractions of hours since midnight. Optional. Numeric expression. If you omit this, the current time is used.

Returns

Formatted date (converted from Julian).

Description

The **DTE()** function converts a date (and time) from Julian form to a formatted string. **fmt\$** defines the format to be returned, in which each component of the date is represented by percent sign (%) followed by a one- or two-letter code. The first letter indicates a source for the data (day, month, year etc.). The second character (if specified) indicates how to format the returned value. The chart below shows the results of the various format options:

% Char1	Source Format	Default	Char2 = l	Char2 = s	Char2 = z
%h	Hour (12)	0-12	0-12	0-12	00-12
%m	Minute	0-59	0-59	0-59	00-59
%p	am or pm	am,pm	am,pm	am,pm	am,pm
%s	Hour in seconds	0-59	0-59	0-59	00-59
%A	Year	00-99, A0-Z9			
%D	Day of month	1-31	Monday-Sunday	Mon-Sun	01-31
%H	Hour (24)	0-24	0-24	0-24	00-24
%J	Day in year	1-365	1-365	1-365	01-99
%M	Month in year	1-12	January-December	Jan-Dec	01-12
%P	AM or PM	AM,PM	AM,PM	AM,PM	AM,PM
%S	Day in seconds	0-86399	0-86399	0-86399	00-86399
%W	Day of week	1-7	Monday-Sunday	Mon-Sun	01-07
%Y	Year	1970-9999	1970-9999	1970-9999	00-99

In general, when the second character is l (*lowercase L*), the result is long text format – an s indicates short text format. ProvideX maintains the exact contents of the text internally. The contents can be changed using a **DEF DTE()** directive. If the second letter is z,

ProvideX supplies the value converted to a two digit value. ProvideX returns a 1-byte binary value if the second letter is p (for compatibility with other languages). The format can also contain YYYY, YY, MM, and DD (e.g., "YYYY/MM/DD") for current *Long Year*, *Year*, *Month* and *Day* values:

Char1	Source Format	Default
DD	Day of month	1-31
MM	Month in year	1-12
YY	Year	00-99
YYYY	Year	1970-9999

If you include any other characters in the date format (e.g., punctuation: slashes, spaces, etc.) ProvideX copies them, as literals, to the output. To include a percent sign as a literal in the output, use %% in the format.

ProvideX includes the **DTE**("%A") format to deal with legacy application Y2K conversions (Version 4.10). The current year is returned using 00-99 for 1900 through 1999, A0-A9 for 2000 through 2009, B0-B9 for 2010 through 2019, etc. ProvideX supports fractional values in the **DTE**() function, using the **DTE**(fraction, *:"form") format.

Examples

```
PRINT DTE(0:"%Dl %Ml %D/%Y %hz:%mz %p")
Monday July 24/1995 10:27 pm

0010 PRINT 'CS',DTE(0:"%Dl %Ml %D/%Y"),@(70),
0010:DTE(0:"%hz:%mz %p")
0020 INPUT "Enter Date (MM/DD/YY):",X$:"00/00/00"
0030 LET M=NUM(X$(1,2))
0040 LET D=NUM(X$(3,2))
0050 LET Y=NUM(X$(5,2))
0060 LET N=JUL(Y,M,D,ERR=0100)
0070 PRINT "That is a ",DTE(N:"%Dl") ! Day value from N, in text, long form.
0080 STOP
0100 PRINT "Invalid date"; GOTO 0010
-:run
Saturday March 27/1999                                03:32 pm
Enter Date (MM/DD/YY):03/31/99
That is a Wednesday
```

The **DTE**() function will support a valid **DAY** formatted string value; e.g.,

```
DAY_FORMAT "MM/DD/AA"
PRINT DTE("01/01/A0":"%Y %Ml %D")
2000 January 1
PRINT DTE("01/01/A0":"%Dz/%Mz/%Y")
01/01/2000
```

See Also

DAY_FORMAT Directive, [p.64](#),
DAY System Variable, [p.557](#)
JUL() Return Julian Date, [p.463](#),
DEF systab= Directives, [p.74](#)
'BY'= System Parameter, [p.658](#)

ENV() Function

Get Environment Values

- Formats
1. *Using Numeric ID*: **ENV**(*env_id*[,*ERR=stmtref*])
 2. *Using String ID*: **ENV**(*env_name*\$[,*ERR=stmtref*])

Where:

- env-id* Numeric value of the environment variable to be returned.
- env_name*\$ Character string containing the environment variable to be returned.
- stmtref* Program line number or label to transfer control to.

Returns Value of an environment variable

Description Use the **ENV()** function to obtain the value of an environment variable (specified numerically or in a string) from the externally defined environment table. Environment variables are typically external to ProvideX and are used by the operating system and other utilities for defining the user's environment.

Format 1: *Using Numeric ID*

ENV(*env_id*[,*ERR=stmtref*])

If *env_id* exceeds the size of the Environment Table, ProvideX returns Error #41: Invalid integer encountered (range error or non-integer). Otherwise, **ENV()** returns the current value of the given environment variable.

The following program displays all the environment variables:

```
0010 LET I=1
0020 LET X$=ENV(I,ERR=0050)
0030 PRINT X$
0040 LET I=I+1; GOTO 0020
0050 STOP
-:run
TEMP=C:\WINDOWS\TEMP
PROMPT=$p$g
winbootdir=C:\WINDOWS
COMSPEC=C:\WINDOWS\COMMAND.COM
CLASSPATH=.;c:\COREL\OFFICE7\SHARED\BARISTA;c:\COREL\OFFICE7\SHARED\TRUEDOC
LD_LIBRARY_PATH=c:\COREL\OFFICE7\SHARED\TRUEDOC\BIN
PATH=C:\WINDOWS;c:\WINDOWS\COMMAND;c:\COREL\OFFICE7\SHARED\TRUEDOC
\BIN;c:\SOFTWARE\WP60;c:\OTHER\BAT;c:\SOFTWARE\WINSIM\CA_APPSW
CMDLINE=WIN
windir=C:\WINDOWS
BLASTER=A220 I5 D1 H5 P330 T6
```


Format 2: *Using String ID*

ENV(env_name\$[,ERR=stmtref])

If *env_name\$* does not match an environment variable, the **ENV()** function returns a null ("") string. Otherwise, the function returns the current value of the given environment variable.

The following program tests and displays the environment variable TERM:

```
0010 T_TYP$=ENV("TERM")
0020 IF T_TYP$="" THEN PRINT "No terminal defined";
0020: STOP
0030 PRINT "You are using a "+T_TYP$+" terminal"
```

EPT() Function

Return Exponent Value

Format `EPT(num[,ERR=stmtref])`

Where:

num Numeric expression whose exponent is to be returned.

stmtref Program line number or label to transfer control to.

Returns Numeric exponent (power of ten).

Description The **EPT()** function returns the power of 10 for the numeric expression provided.

Examples

```
->EPT(10)    ! Same as EPT(0.1*10^2)   - yields 2  
->EPT(5)    ! Same as EPT(0.5*10^1)   - yields 1  
->EPT(.02) ! Same as EPT(0.2*10^-1) - yields -1
```

```
0610 LET B=27*9  
0620 PRINT EPT(B), EPT(9*2)  
-:run  
3 2
```

ERR() Function

Test Error Value

- Format**
1. *Return ERR Position*: **ERR(compare1,compare2,...[,ERR=stmtref])**
 2. *Return Extended Error Information*: **ERR(keyword\$ | *)**

Where:

- * *Asterisk*. Lists the names of the returned values.
- compare1, compare2, ...* Comma-separated list of numeric values for comparison with the value of the internal **ERR** variable.
- keyword\$* Currently supported return values:
 - "ERR" Error condition
 - "RET" OS error
 - "OSERR" OS error message; i.e., **MSG(-1)**
 - "PROGRAM" Pathname of program with error
 - "STNO" Statement number
 - "OBJ" Object number
 - "METHOD" Method name invoked
 - "LFA" Last file accessed
 - "LFO" Last file opened
 - "LASTPATH" Last pathname referenced
 - "LASTKEY" Record key value, only valid for Error 11's.
- stmtref* Program line number or label to transfer control to.

Returns Integer, relative position if match found, 0 *zero* if no match found.

Description The **ERR()** function can be used to determine the value of the internal **ERR** variable or receive additional information about the last un-trapped error.

Format 1: Return ERR Position

ERR(compare_1,compare_2,...compare_n[,ERR=stmtref])

Use this format to compare the value of the internal **ERR** variable with a list of possible external error values. If a match is found, the function returns an integer reporting the relative position of the value which matches the value in the external **ERR** system variable. The function returns 0 *zero* if no match is found.

Example:

```
0040 ON ERR(1,11,2) GOTO 1000,1010,1110,1020
```

Possible results:

- ERR=1 Control transfers to line 1010
- ERR=11 Control transfers to line 1110
- ERR=2 Control transfers to line 1020
- ERR=0 *No match*. Control transfers to line 1000.

Format 2: Return Extended Error Information**ERR(keyword\$)**

This format provides additional information for diagnosing untrapped errors. The information returned by the **ERR()** function is not affected by errors that are programmatically trapped using a **SETERR** or any of the **ERR=/DOM=/BSY=** options.

Example:

```

0010 ! Display ERR(" ") return values
0020 error_handler pgn+";ErrorHandler"
0030 print 4/0
0040 stop
0050 !
0100 ! ^100
0110 ErrorHandler:
0120 print "Un-trapped error",err,":"
0130 x$=ERR(" *")
0140 !
0200 ! ^100
0210 x=pos(", "=x$); if x=0 then exit err
0220 print pad(x$(1,x-1),12), "= ",
0230 print ERR(x$(1,x-1))
0240 x$=x$(x+1); goto 0200

```

See Also

[ERR System Variable, p.560.](#)
[Error Codes and Messages, p.828](#)
[User's Guide, Chapter 3](#)

EVN() Function Evaluate Numeric Expression

Format **EVN(var\$[,val][,ERR=stmtref])**

Where:

stmtref Program line number or label to transfer control to.

val Default *value* to be returned if the evaluation fails at run-time (with any error except a syntactical error). If supplied, the error will be ignored and the value will be returned instead. For instance, `PRINT EVN("40/0" , 0)` would return *val*=0 rather than generating `Error #40: Divide check or numeric overflow`.

var\$ String expression containing numeric variable name. Maximum string size 8kb. Receives the returned evaluated contents of the variable. You can build the variable name using string expressions, as in the example below.

Returns Processed numeric value.

Description The **EVN()** function evaluates and returns the numeric value of a numeric variable or computed expression. Use this function to process a stored or computed expression and obtain its value.

See Also [EVS\(\) Evaluate String Expression, p.430](#)
[VIN\(\) and VIS\(\) Obtain Value of Variable, p.549](#)

Examples In this example, `EVN("GL_" +X$)` builds the variable name based on user input and returns the value stored in the variable.

```
00010 GL_YTD=10000,GL_MTD=3000,GL_CUR=10
00020 input "Which field (YTD,MTD,CUR):",X$
00030 print evn("GL_" +X$):"$###,##0.00-"
00040 stop
```

For the example above, if the user's input is CUR for X\$, the variable name will be GL_CUR. If the value stored in GL_CUR in the current record is, for example, 9999.63, ProvideX prints that value (using the format mask) as \$9,999.63.

In this example, `EVN(V$)` will print the numeric value stored in V\$ for the current record of logical file number RPT_FN.

```
0090 K$=KEY(RPT_FN); IF K$(1,12)<>RPT_ID RETURN
0100 READ (RPT_FN)L,C,V$
0110 PRINT (RPT_FN)@(C,L),EVN(V$)
0120 GOTO 0090
```

EVS() Function*Evaluate String Expression*

Format **EVS(var\$[,val\$][,ERR=stmtref])**

Where:

stmtref Program line number or label to transfer control to.

val\$ Value to be returned if the evaluation fails at run-time (with any error except a syntactical error). Optional. If you include a value, the error will be ignored and the value will be returned instead; e.g.,
A=9999; PRINT EVS("str(a: "##0 ")", "<Overflow>")
returns the value <Overflow> rather than generate
Error #43: Format mask invalid.

var\$ String variable name. Receives the returned evaluated contents of the variable. You can build the variable name using string expressions, as in the example below.

Returns Evaluated contents of string variable.

Description The **EVS()** function evaluates and returns the value (evaluated contents) of a string variable.

See Also [EVN\(\) Evaluate Numeric Expression, p.429](#)
[VIN\(\) and VIS\(\) Obtain Value of Variable, p.549.](#)

Examples In this example, **EVS()** builds the string variable's field name ("CST_" + X\$ + "\$") based on the user's input and returns the value stored in it.

```
00010 CST_NAME$=who,CST_ADDR$="8920 Woodbine Ave"
00020 CST_CITY$="Markham",CST_CONTACT$="Support"
00030 input "Which field (NAME,ADDR,CITY,CONTACT):",X$
00040 print evs("CST_" + X$ + "$")
00050 stop
```

For instance, if the user's input for X\$ in the example above is NAME, the variable name will be CST_ADDR\$. The value in EVS(CST_ADDR\$) for the current record could be *8920 Woodbine Ave*, which is what ProvideX would return and then print.

EXP() Function

Raise to Base Ten

Format `EXP(num[,ERR=stmtref])`

Where:

num Power of 10 (ten) to be returned. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Value of ten raised to the power of the numeric value

Description The **EXP()** function returns the value of 10 raised to the power of a given numeric value (i.e., 10^{num}). Fractions are allowed. The numeric value returned is rounded to the current **PRECISION** in effect. This is the inverse of the **LOG()** function.

Examples

```
0010 PRINT EXP(LOG(90)/3), ! Print cube root of 90
0020 PRINT EXP(3),EXP(2.1)
-> RUN
4.47 1000 125.89
```

FFN() Function

Find File Number

Format **FFN(filename\$[FOR OBJECT] [,ERR=stmtref])**

Where:

filename\$ Name of the file to locate. String expression (case sensitive on UNIX/Linux systems.)

FOR OBJECT Keyword to return file handles owned by an object. Inside an object the first reference to the file owned by the object is returned otherwise it will be -1.

stmtref Program line number or label to transfer control to.

Returns Integer, channel/file number for open file (-1 if not open).

Description This function checks all open files for any reference to the *filename\$* specified. The function returns the file number as an integer if the file is open. If the file is not currently open on any channel, **FFN()** returns a value of -1. The filename specified in **FFN()** must match the name used to **OPEN** the file; e.g.,

```
open(1)"cstfile
print "channel is: ",ffn("C:/Program Files/Sage
    Software/ProvideX/NOMADS/cstfile") ! Incorrect
channel is: -1
print "channel is: ",ffn("cstfile") ! Correct
channel is: 1
```

Examples Read a record from the CUSTDB file. If the file is not open, then open it on a Global file number.

```
0100 LET X=FFN("CUSTDB")
0110 IF X=-1 THEN LET X=GFN; OPEN (X)"CUSTDB"
0120 READ (X,KEY=CST_ID$,ERR=9000)IOL=%CST_IOL$
```

FFN() is case-sensitive (as is necessary in UNIX). To perform a case-insensitive search for the filename, set either '**FL**' (lower case) or '**FU**' (upper case) parameters.

```
0100 OPEN (5)"CSTfile" ! With neither 'FL' nor 'FU' set
-:PRINT "channel is: ",FFN("CSTFILE")
channel is: -1
-:SET_PARAM 'FL'
-:PRINT "channel is: ",FFN("CSTFILE")
channel is: 5
-:PRINT "channel is: ",FFN("cstFILE")
channel is: 5
```


Forward slashes may be used in place of backslashes.

```
open(1)"C:\Program Files\Sage Software\ProvideX\NOMADS\cstfile"  
print "channel is: ",ffn("C:/Program Files/Sage  
    Software/ProvideX/NOMADS/cstfile")  
channel is: 1
```

A leading ./ or .\ is ignored by the **FFN()** function.

```
open(1)"/cstfile  
print "channel is: ",ffn("cstfile")  
channel is: 1
```

FIB() Function

Return File Information Block

Format **FIB**(*chan* [, **ERR**=*stmtref*])

Where:

chan Channel/logical file number of the file about which you want information.

stmtref Program line number or label to transfer control to.

Returns String, file information block description.

Description The **FIB**() function returns a character string containing a file information block description for an existing open file. This information may be passed to a **FILE** directive to subsequently recreate the file. Refer to the [File Information Functions Overview](#) and the [ProvideX Standard Format for FIB\(0\) and FID\(0\)](#) for further details. Unlike **FID**() , the **FIB**() function is not affected by emulation mode settings.



Note: When 'FF' is set to 0 or 3 and the 'PO' system parameter is switched *on*, the **FID**() and **FIB**() functions return the original path used when the file was opened. 'FF'.

Examples

```
0010 INPUT "Enter filename:",F$
0020 OPEN (1)F$
0030 IF MID(FIB(1),10,1) <> $02$ THEN PRINT "Not a Keyed file"
```

See Also

[FID\(\) Return File Information Descriptor, p.438](#),
[FIN\(\) Return File Information, p.441](#),
[FILE Directive, p.130](#)
['FF' File format, p.666](#)
['PO' Path Original, p.680](#)

File Information Functions Overview

The **FIB**() , **FID**() and **FIN**() functions return file information, descriptors and blocks.

FID() returns file characteristics or descriptors such as the type of file, size, keys (if applicable) and the absolute location (pathname) of the file on the disk system. **FIB**() always returns the information in ProvideX Standard Format (described in the following section).

ProvideX returns file identifiers for a file through either the **FID**() or **FIB**() function. Normally these two functions will both provide the same information, but to simplify conversion to ProvideX from various Business Basics, ProvideX returns one of 5 different formats for the **FID**() function, based on the setting of the 'FF' system parameter.

There is *one exception* to the above. For a device, the **FID**() function always returns only the device name. For example, **FID**(0) will yield the terminal identifier.

You can define the value returned by **FID(0)** externally using the Environment variable **PVXFID0** or internally using the **SETFID** directive. **FIN()** returns file information about the data structure (describing the physical aspects of a file such as maximum record, number of records, key size, record size).

ProvideX Standard Format for **FIB(0)** and **FID(0)**

The following charts describe the **FID(0)/FIB(0)** ProvideX standard format for 'FF'=0.

Bytes	Description (Total length = 212 bytes)
1,3	Current record count (if available)
4,6	Characters 1-6 of filename (will include the path, if used)
10,1	\$00\$ - Indexed \$01\$ - Serial \$02\$ - Keyed \$03\$ - Open via ISZ= (binary) \$04\$ - Program \$05\$ - Attached printer \$07\$ - Isam \$0A\$ - Pipe \$0D\$ - Directory \$0E\$ - TCP/IP \$0F\$ - Device or Windows printer
11,1	External key size
12,3	Maximum number of records
15,2	Record size
17,1	Keyed file flag (\$01\$ if variable length records)
18,1	Keyed file inventory threshold %
19,1	File type: "*" Device " >" I/O redirection "2" EFF file type indicator "A" Attached printer "B" Opened via ISZ= "b" DB2 file "C" C-ISAM "D" Directory "d" BBx Directory "E" DDE Connection "F" PDF File "I" Indexed "K" Keyed/Direct/Sort "l" DLL file "M" Dynamic * MEMORY * File "m" BBx M-Keyed File "N" TCP/IP Connection "o" OCI Data file "O" ODBC Data file "P" Program "p" Program Library Pseudo File (<i>internal use only</i>) "S" Serial "T" WindX Connected Device/File "V" Variable Data File "W" Windows Printer "Y" * MEMORY * file with alternate keys.
20,1	External file handle
21,1	Current attribute byte

Bytes	Description (Total length = 212 bytes)
22,1	Current foreground colour
23,1	Current background colour
24,1	Reserved for future use
25,60	External file pathname

ProvideX Standard FIB(0) / FID(0) 'FF'=0 for Keyed Files

Bytes	Description FIB(0) / FID(0)
85,384	<p>4 or 8-byte extended entries (for extended key attributes). The maximum number of key segments can range between 48 and 96 depending on the number of 4 and 8 byte entries.</p> <p>(1,1) Key number in four low order bits; e.g., $KEYNUM=DEC(AND(KDAT\$(1,1), \$0F\$))$ Four high order bits contain offset information used in conjunction with byte (3,1). \$FF\$ indicates final entry. \$FE\$ indicates it is byte (5,1) of an extended entry</p> <p>Note: When working with an EFF file, the key number will be reset to zero after the 16th key. For example (a file with 35 keys) byte (1,1) will go from 0 to 15 then for key 17 byte (1,1) will be zero again. It will climb to 15 then for the 33rd key byte (1,1) will be zero again. Byte (1,1) for the 35th key will be 2.</p> <p>(2,1) Field # in record ($FF\$=KEY$)</p> <p>(3,1) Offset in field (in conjunction high order four bits of byte (1,1)), maximum 3839; e.g., $KEYOFFSET=16*DEC(\$00\$+AND(\$F0\$, KDAT\$(1,1)))+DEC(\$00\$+KDAT\$(3,1))$</p> <p>(4,1) Length of key (descending if bit \$80\$ is on)</p> <p>Extended entry:</p> <p>(5,1) \$FE\$ indicates extended entry, otherwise it is byte (1,1) of next entry.</p>

Bytes	Description FIB(0) / FID(0)
	(6,2) Segment attributes (cumulative): \$0001\$ Unique key \$0002\$ Convert segment to upper case \$0004\$ Convert segment to lower case \$0008\$ Convert using translate table \$0010\$ Swap byte order \$0020\$ Primary key allows duplicates \$0040\$ Don't add key if the segment is null \$0100\$ Don't add key if all segments are null \$0200\$ Binary auto-increment key \$0400\$ Ignore data after \$00\$ \$0800\$ Zero-filled auto-increment \$1000\$ Space-filled auto-increment (8,1) Null character for NULL keys ((6,2)=\$0040\$ or\$0100\$)

ProvideX Standard FIB(0) / FID(0) 'FF' = 0 for Terminal Devices:

Bytes	Description FIB(0) / FID(0)
85,1	Current column
86,1	Current line
87,1	Maximum column
88,1	Maximum line
89,110	Window information (10 * 11 byte entries for top 10 windows): (1,1) Window number (2,2) Absolute column (2 Byte) (4,2) Absolute line (2 Bytes) (6,1) Number of columns wide (7,1) Number of lines high (8,1) Column 0 of <i>scroll</i> region (9,1) Line 0 of <i>scroll</i> region (10,1) # columns for <i>scroll</i> region (11,1) # lines for <i>scroll</i> region
213,12	Actual device type

FID() Function Return File Information Descriptor

Format FID(*chan*[,ERR=*stmtref*])

Where:

chan Channel or logical file number of the file to return information about.

stmtref Program line number or label to transfer control to.

Returns String, file information descriptor.

Description The FID() function returns a character string containing the file information descriptor for an existing open file. If the file is a device (printer, terminal, etc.), the filename used to open the file will be returned. If the file is a disk file, a file description string is returned. You can make subsequent use of this file description by passing it to a **FILE** directive to recreate the file. For further information, refer to the [File Information Functions Overview, p.434](#) and the [ProvideX Standard Format for FIB\(0\) and FID\(0\), p.435](#).

If you are running ProvideX in an emulation mode, the format of the information returned will be changed to reflect the system being emulated.



Note: When 'FF' is set to 0 or 3 and the 'PO' system parameter is switched *on*, the FID() and FIB() functions return the original path used when the file was opened.

Also: The FID() and FIN() format layouts will be changed whenever there is a change to the 'FF' system parameter.

Determining FID in ProvideX under UNIX

By default, ProvideX assigns a unique FID value for each terminal based on the /etc/inittab entry for the terminal. Unfortunately, when you use dynamic terminal allocation (e.g., Telnet), each time a user signs off and then signs on, he / she will have a different FID value. There are workarounds, but no one solution works in all instances. If, for security purposes, you use the `user_ID` rather than `terminal_ID`, then you can set the file information descriptor in the user's profile by inserting the following lines:

```
PVXFID0=desired_FID_val
export PVXFID0
```

As an alternative, you can assign the file information descriptor once in ProvideX, by issuing a **SETFID** directive to establish a new **FID(0)** value. If you do not want to use `user_IDs` but you are using Telnet and PCs, see if there is a host-initiated file transfer you could use to get a file containing the FID value off the PC.

Using WindX, you could open a file on the PC and read it to determine a FID value; e.g.,

```
0010 OPEN (1) "[wdx]C:\PVXFID"
0020 READ RECORD (1) F$
0030 CLOSE (1)
0040 SETFID F$
```

Finally, you could see if the terminal emulator allows you to define values for function keys and has an escape sequence to send the values. Some terminals send a "HEREIS" string on receipt of an ENQ (hex \$05\$), but this varies from terminal to terminal.

FIB(1) / FID(1) 'FF' =1

Bytes	Description (Total length=22 bytes)
1,3	\$000000\$
4,6	Characters 1-6 of filename
10,1	File type indicator: \$00\$ - Indexed \$01\$ - Serial - WindX connected file (any type) \$02\$ - Keyed or EFF \$03\$ - Open via ISZ= (binary) \$04\$ - Program \$0D\$ - Directory \$0F\$ - Device
11,1	External key size
12,3	Maximum number of records
15,2	Record size
17,4	Reserved for future use
21,2	Characters 7-8 of filename

FIB(2) / FID(2) 'FF' =2

Bytes	Description (Total length=16 bytes)
1,6	Characters 1-6 of filename
7,1	File type indicator: \$00\$ - Indexed \$01\$ - Serial - WindX connected file (any type) \$02\$ - Keyed \$03\$ - Open via ISZ= (binary) \$04\$ - Program \$0D\$ - Directory \$0F\$ - Device
8,1	External key size
9,2	Maximum number of records (bytes reversed)
11,2	Record size (bytes reversed)
13,4	Not used

FIB(3) / FID(3) 'FF' = 3

Bytes	Description (Total length=9+ bytes)
1,1	File type indicator: \$00\$ = Indexed \$01\$ = Sequential - WindX connected file (any type) \$02\$ = Keyed \$03\$ = Opened via ISZ = (Binary) \$04\$ = Program \$05\$ = Directory \$06\$ = BBx Mkeyed (no External Key) \$07\$ = C-Isam \$0A\$ = UNIX Pipe \$0E\$ = TCP/IP-Network \$0F\$ = Device/Attached Printer
2,1	External key size
3,4	Maximum number of records
7,2	Record size (bytes reversed)
9,...	Pathname; e.g., 0100 SET_PARAM 'FF'=3 0110 OPEN (30)"SESAME" 0120 LET A\$=FID(30) 0130 PRINT A\$(9) ->run C:\OTHER\TESTS\SESAME
1,...	Device name; e.g., 0200 SET_PARAM 'FF'=3 0210 OPEN (31)PRINTER\$ 0220 B\$=FID(31); PRINT B\$ ->run LPT1 [Details for files, above (1,1 to 9,...) inapplicable]

FIB(4) / FID(4) 'FF' = 4

FID(4) is the same as **FID(3)** but all serial files return a file type indicator of \$03\$ for binary. This format is only intended to facilitate program migration to ProvideX.

See Also

FIB() Return File Information Block, p.434,
FIN() Return File Information, p.441,
FILE Directive, p.130,
'FF' File format, p.666
'PO' Path Original, p.680

FIN() Function

Return File Information

Formats

1. *Return File Information*: **FIN**(chan[,ERR=stmtref])
2. *Return Detailed Information*: **FIN**(chan,field\$[,ERR=stmtref])

Where:

- chan* Channel or logical file number of the file to return information about.
stmtref Program line number or label to transfer control to.
field\$ String expressions (case insensitive). For valid keywords, refer to the description of Format 2 below.

Returns

String, physical information about open file.

Description

The **FIN()** function returns a character string containing details about an existing open file. The information returned is not the same as that in the **FIB()** function. While some of the information in the **FIN()** function is common to information in the **FIB()** function, the **FIN()** function includes more detailed information about the physical file (e.g., file size in bytes, date and time of last update, etc.).



Note: The **FID()** and **FIN()** format layouts will be changed whenever there is a change to the 'FF' system parameter.

Format 1: Return File Information

FIN(chan[,ERR=stmtref])

In the example below, the **FIN()** function is used to find out the number of characters in *IN_FILE\$* (the variable contains the name of a serial file).

```
0140 LET IN_FILE=HFN; OPEN (IN_FILE)IN_FILE$
0150 LET F$=FIN(IN_FILE)
0160 CLOSE (IN_FILE)
0170 LET CHARS=DEC(F$(1,4))
0180 !
```

Format 2: Return Detailed Information

FIN(chan,field\$[,ERR=stmtref])

Use this format to have the **FIN()** function return details. For example, **FIN**(10, "DEVHANDLE") returns the value of the Windows API Device handle for a communications port. The following keywords, as well as '**OPTION**' keywords, are valid for use with the **FIN()**:

- BUFFERED** - Returns 1 for output buffered, 0 for not buffered.
- CURKNO** - Returns the current KNO value
- DEVHANDLE** - Windows API device handle for communications port
- DRIVER** - ProvideX device driver name
- EXTRACT** - Extract status of file (1 if Extract pending)
- FILE_CREATE** - Returns file creation command for any ProvideX-based file.

FILELENGTH - Size of physical file.
FILENAME - Original filename used in **OPEN**.
HANDLE - Operating system file handle.
HDC - For ***WINDEV*** channel, returns handle as returned by the Windows API call `OpenPrinter()`. Otherwise, returns handle to the device context for ***WINPRT*** channel.
IPADDR TCP/IP - Address.
IO_PROGRAM - Program filename of imbedded I/O.
KEY_DEFINITION - Human readable key definition.
KEY_NAMES - List of Named Keys for an open channel.
KEY_OPTIONS - Returns the **OPT=** value.
KEY_SIZE - External Key size of a Keyed file.
KSZ - Same as **KEY_SIZE**.
MAXKNO - Maximum file access key number allowable for the file.
MAXREC - Same of **RECORDS_ALLOWED**.
NAME - Same as **PATHNAME**.
NUMREC - Number of records used.



Note: The record count returned by **NUMREC / RECORDS_USED** is based on the last file I/O operation performed by the current task. To obtain up-to-date values, force an I/O operation against the file prior to requesting this information, or use **FIB()** instead.

PATHNAME - Full pathname of file.
RECORD_SIZE - Maximum record size.
RECORDS_ALLOWED - Maximum # of records.
RECORDS_USED - Same as **NUMREC**.
RSZ - Same as **RECORD_SIZE**
SEPARATOR - Field separator character or **"*Dynamic*"**.
SSL_CIPHER - SSL connection cipher used information as per the OPENSSL specs.
SOURCELIST - Provides list of available printer source trays (*tray number:name*).
TRUE_TTY - UNIX/Linux only. Returns FacetTerm TTY
UTC_CTime - Creation/changed time, UTC in seconds since Jan 1,1970
UTC_MTime - Last modified time, UTC in seconds since Jan 1,1970.
UTC_ATime - Last accessed time, UTC in seconds since Jan 1,1970.



Note: **UTC_CTime**, **UTC_MTime**, and **UTC_ATime** (Universal Coordinated Time) values returned are OS dependant. Exact definitions of these items may only be determined by checking the OS/file system documentation including the characteristics of the `stat` function.

X509_ISSUER - Who issued SSL certificate.
X509_NOT_AFTER - Latest date the certificate is valid for.
X509_NOT_BEFORE - Earliest date certificate is valid for.
X509_KEYTYPE - What type of key is in certificate.
X509_SUBJECT - Who certificate is issued to.



Note: All the **X509_** keywords return information pertaining to a remote SSL server certificate (if a client requests it) or client certificate (if the server requests it). When on a server, the information pertains to the last/current socket connected. **X509_** requests can also be prefixed with **MY_** to return the local station's certificate information (e.g., **MY_X509_KEYTYPE**).

Keywords for the 'OPTION' Mnemonic

As mentioned earlier, keywords used in the 'OPTION' mnemonic can also be used by the **FIN()** function; e.g, **COLOURnnn**, **FONT**, **ICON**, etc. For the complete list of available 'OPTION' keywords, refer to the 'OPTION' Mnemonic, p.624.

For UNIX FacetTerm Use Only

The **FIN()** function reports the actual TTY on which the UNIX FacetTerm session was initiated. If the environment variable `FACETTYPE` is set / ON in a current FacetTerm session, recognition is automatic. `FIN(0, "TRUE_TTY")` will return the `TRUE /dev/tty` terminal or device.

The value returned on a system without FacetTerm will be the same as the value returned by **PTH(0)**. For more information, see **PTH() Return Pathname, p.506**.

FIN() For Disk Resident Files

Bytes	Description for Disk Resident Files (Total length=68 bytes) FIN()
1,4	Number of bytes in the file
5,4	Date/Time of last modification (Seconds since Jan. 1, 1970)
9,4	Date/Time of last access
13,4	Date/Time of file creation/change
17,2	Physical device number
19,4	Inode number (UNIX/Linux only)
23,2	UserID of file owner (UNIX/Linux only)
25,2	GroupID of file owner (UNIX/Linux only)
27,2	Status flag. Bitmapped values for OPEN clauses: \$0001\$ - READ Ok \$0002\$ - WRITE Ok \$0004\$ - EXECUTE Ok (<i>Windows only flag value</i>) \$0008\$ - Is a directory \$0010\$ - LOCK 'ed \$0020\$ - OPEN INPUT \$0040\$ - OPEN LOAD Bitmapped value for PURGE clause: \$0080\$ - File has been purged.
60,4	Maximum record count
64,4	Current record index



Note: File times returned by the **FIN()** function on UNIX/Linux systems are reported based on the current time zone, rather than GMT. This keeps the file time reporting consistent with Windows versions of ProvideX.

FIN() For Disk Resident Keyed Files Only.

Bytes	Description for Disk Resident Keyed Files Only FIN()
77,4	Number of records on file
85,1	Current access key number
86,128	Key definition data. See FIB() and FID() for Keyed Files, bytes 85, 384 or use utilities to return key information (e.g., *UFI and **KEY.INF)

*FIN()* For Device Files only

Bytes	For Device Files Only. (TCP Files are Reported as Devices.) FIN()
1,1	Current column
2,1	Current line number
3,1	Number of columns
4,1	Maximum number of lines.
5,1	Column offset to start of scroll region
6,1	Line offset to start of scroll region
7,1	Current width of scroll region
8,1	Current height of scroll region
9,1	Reserved (always \$00\$)
10,1	Current window number
11,1	Reserved (always \$00\$)
12,1	Current device attribute bits
13,1	Current foreground colour
14,1	Current background colour
15,1	Reserved (always \$00\$)
16,1	Default attributes
17,1	Default foreground colour
18,1	Default background colour
19,2	Current device mode
21,4	Current device status words 1 and 2
25,2	Auxiliary attributes
27,2	Device option flags
29,2	Operating system handle (Windows only)
31,1	Standard character width
32,1	Standard character height
33,...	Pathname to device (\$00\$-terminated)
33+n...	Device type (\$00\$-terminated)

See Also

[FIB \(\) Return File Information Block, p.434,](#)
[FID \(\) Return File Information Descriptor, p.438,](#)
['OPTION' Mnemonic, p.624](#)
[FILE Directive, p.130,](#)
['FF'= System Parameter, p.666](#)
['PO' System Parameter, p.680](#)

FPT() Function

Return Fractional Part

Format `FPT(num[,ERR=stmtref])`

Where:

num Numeric expression whose fractional portion will be returned.

stmtref Program line number or label to transfer control to.

Returns Fractional portion of numeric.

Description The **FPT()** function returns the fractional portion of the given numeric value. The value returned is *always* rounded to the **PRECISION** currently in effect.

Examples

0020 PRECISION 3	Set precision
0030 ...	
0100 LET A=FPT(1.345)	yields A = .345
0110 LET B=FPT(A/10)	yields B = .035
0040 BEGIN	Set precision to 2
0050 LET A=FPT(5/3)	yields A = .67

GAP() Function

Return Odd Parity String

Format **GAP(string\$[,ERR=stmtref])**

Where:

string\$ Character string to convert to Odd parity.

stmtref Program line number or label to transfer control to.

Returns Odd parity value of string

Description The **GAP()** function converts and returns the character string expression in Odd parity. This function is typically used when dealing with communication lines. It is not normally used in operating systems since the standard communications drivers handle the generation of parity.

The number of one-bits in each byte of the character string determines the parity of data. Odd parity data always has an odd number of one-bits in each byte of data.

See Also [GEP\(\) Return Even Parity String, p.449](#)

Examples

```
0110 PRINT "HTA(""This is a test"")      =",HTA("This is a test")
0120 PRINT "HTA(GAP(""This is a test""))=",HTA(GAP("This is a test"))
-:run
HTA("This is a test")      =5468697320697320612074657374
HTA(GAP("This is a test"))=5468E97320E973206120F4E573F4
```

GBL() Function Reference Global String Variable

Formats

1. *Get / Maintain Entries*: **GBL**(string_name\$[,contents\$][,ERR=stmtref])
2. *Delete / List Single Value*: **GBL**({DELETE | LIST}string_name\$[,ERR=stmtref])
3. *Delete / List All up to Value*: **GBL**({DELETE | LIST} TO string_name\$[,ERR=stmtref])
4. *Delete / List Table*: **GBL**({DELETE | LIST} *[,ERR=stmtref])

Where:

* Asterisk. Indicates *all* global string table entries.

contents\$ Value to assign to the global string. Optional. String expression.

stmtref Program line number or label to transfer control to.

string_name\$ Name of the global string in the internal table. String expression.

Returns

Values in internal table of strings.



Note: This function is primarily provided for compatibility with other languages and has been made virtually obsolete by global variables – a much more efficient way to handle common data elements (string, numeric, arrays, etc.).

Description

The **GBL()** function returns and maintains the values in an internal table of string definitions.

Format 1: *Get / Maintain Entries*

GBL(string_name\$[,contents\$][,ERR=stmtref])

Use this format to set or obtain the current value of the global string variable whose name you specify. If you include a contents value (optional) it will be placed into the global string variable.

Format 2: *Delete / List Single Value*

GBL({DELETE | LIST}string_name\$[,ERR=stmtref])

Use the **GBL(DELETE ...)** format to delete an individual global string from the table.
Use the **GBL(LIST ...)** format to have the function return an individual string.

Format 3: Delete / List All up to Value**GBL({DELETE | LIST} TO *string_name*\$[,ERR=*stmtref*])**

Use the **GBL(DELETE TO ...)** format to delete items up **TO** an individual global string from the table. Use the **GBL(LIST TO ...)** format to have the function return a list of global string names up **TO** an individual item (\$00\$ separated).

Format 4: Delete or List Table**GBL({DELETE | LIST} *[,ERR=*stmtref*])**

Use the **GBL(DELETE *)** format to remove all entries from the table. The **GBL(LIST *)** format returns the names of all strings in the table (\$00\$ separated).

GEP() Function

Return Even Parity String

Format **GEP(string\$[,ERR=stmtref])**

Where:

string\$ Character string to convert to Even parity.

stmtref Program line number or label to transfer control to.

Returns Even parity value of string.

Description The **GEP()** function converts and returns the character string expression in Even parity. This function is typically used when dealing with communication lines. It's not normally used by the operating system since the standard communications drivers handle the generation of parity.

The number of one-bits in each byte of the character string determines the parity of data. Even parity data always has an even number of one-bits in each byte of data.

See Also [GAP\(\) Return Odd Parity String, p.446](#)

Examples

```
0110 PRINT "HTA("This is a test")      =",HTA("This is a test")
0120 PRINT "HTA(GEP("This is a test"))=",HTA(GEP("This is a test"))
-:run
HTA("This is a test")      =5468697320697320612074657374
HTA(GEP("This is a test"))=D4E869F3A069F3A0E1A07465F374
```

HSA() Function

Highest Sector Available

Format `HSA(drive_num[,ERR=stmtref])`

Where:

`drive_num` Numeric value representing the disc drive number.

`stmtref` Program line number or label to transfer control to.

Returns Always 0 zero.



Note: Included for compatibility with other languages.

Description ProvideX includes the **HSA()** function for compatibility with Business Basic languages where it returns the highest sector number available on the disk drive specified. Since this function is not applicable in ProvideX, it always returns 0 zero.

HSH() Function

Generate Modified Value

Format

1. *Hash Value*: **HSH**(string\$[,Hkey\$][,Htype [,KeyHashedWith]][,ERR=stmtref])
2. *Encrypted Value*: **HSH**(PASSWORD string\$ WITH cipher\$[,options][,ERR=stmtref])
3. *Decrypted Value*: **HSH**(EXTRACT string\$ WITH cipher\$[,options][,ERR=stmtref])

Where:

<i>cipher\$</i>	Encryption/decryption method. See Recognized Methods, p.453 .
EXTRACT	Keyword indicating <i>decryption</i> .
<i>options</i>	Supported encryption/decryption options: KEY=string\$ Key value to encrypt / decrypt the data with SIZ=num Optional value to specify/override total length of key value. If greater than length, key value will be padded with trailing null characters. TBL=value\$ Optional initialization value used by some ciphers
PASSWORD	Keyword indicating <i>encryption</i> .
<i>string\$</i>	String expression whose value is to be returned.
<i>Hkey\$</i>	String expression representing key to use during the hashing operation. See Hash Types, p.451 .
<i>Htype</i>	Optional numeric value representing the type of hash to return for the data. See Hash Types, p.451 .
<i>KeyHashedWith</i>	Optional numeric value (for <i>Htype 7</i>) used to specify which <i>Htype</i> the <i>Hkey\$</i> is based on. See Hash Types, p.451 .
<i>stmtref</i>	Program line number or label to transfer control to.

Returns

Modified string value for the given data.

Description

Depending on the format used, the **HSH()** function can return a hash value or a encrypted/decrypted value for a given string.

Format 1: Return Hash Value

HSH(string\$[,Hkey\$][,Htype [,KeyHashedWith]][,ERR=stmtref])

This format returns a *hash value* for a given string. The hash value returned in a two-byte string can be used to check the integrity of a character string. The initial value can be used to calculate the hash value of an entire string by taking its component parts.

Hash Types

An optional numeric value (*Htype*) can be applied to represent the type of hash to return for the data. Supported hash types are defined by the associated *Htype* values listed below.

- 0 ProvideX 2-byte hash (default if not specified)
- 1 MD5
- 2 MD4
- 3 MD2
- 4 SHA1
- 5 MDC2
- 6 RIPEMD
- 7 HMAC

An invalid value causes `Error #41 Invalid integer encountered`. **0** is the internal hash and is always available. If a hash type is not specified, **0** is assumed.

If *Htype* is from **1** to **7**, OpenSSL libraries are required to perform the hash. Only versions of ProvideX that support OpenSSL and have OpenSSL installed properly will be able to access these hashes. The hash type must also exist within the OpenSSL modules for the extended hash types to be available. Not all builds of OpenSSL contain all possible hashes. If a specific hash type is not available, an `Error #99: Feature not supported` is reported.

If *Htype* is **7** (HMAC), a key value (*Hkey\$*) must be supplied for the hashing operation. This must be 2 characters in length or an `Error #46: Length of string invalid` is generated. *Hkey\$* is optional for when *Htype* is **0** and it is ignored for *Htype* **1** through **6**.

If *Htype* is **7** (HMAC), a numeric value (*KeyHashedWith*) can be used to specify which *Htype* the *Hkey\$* is based on (values **0** to **6**). This only applies to *Htype* **7**. The HMAC hash is a special case. Data that has been hashed with a *Htype* such as MD5 will return an MD5 hash key. When the original data and the MD5 hash key are hashed together as an HMAC, the new HMAC hash is called a *Message Authentication Code*. An invalid value results in an `Error #41: Invalid integer encountered`.

Examples

To get a ProvideX hash of a string:

```
PRINT hta(hsh("An internal ProvideX Hash"))
83C9
```

To get a ProvideX hash of a string based on a key:

```
PRINT hta(hsh("An internal ProvideX Hash based on a Key", "K1", 0))
FCC2
```

To get an MD5 hash:

```
PRINT hta(hsh("A string to be MD5 hashed", 1))
C9755C05F3EF1704114446A04F4072DF
```

To get or check a *Message for Authentication* based on HMAC-SHA1:

```
Data$="This is a string of data"
SHA1Hash$ = HSH(Data$, 4)
MessageAuthenticationKey$ = HSH(Data$, SHA1Hash$, 7, 4)
IF KeyReceived$ <> MessageAuthenticationKey$
  THEN MSGBOX "Message has been tampered with"
```

Formats 2 and 3: Return Encrypted or Decrypted Value

HSH(PASSWORD|EXTRACT *string*\$ WITH *cypher*\$[,*fileopt*][,ERR=*stmtref*])

This format returns a string expression to be *encrypted* or *decrypted* based on an encryption/decryption method (*cypher*\$ value). If *string*\$ is an *astrik* and *cypher*\$ is not supplied, then a comma-separated list of available encryption/decryption methods is returned.

Recognized Methods

Encryption/decryption methods (*cypher*\$ values) are supported by ProvideX; however, the OpenSSL libraries used may not have been created with all the available algorithms. If the OpenSSL libraries do not have the requested function, the result will be an **Error #99: Feature not supported**. In addition, it is up to the software developer to understand how to use any particular algorithm.

Bf-cbc	DES-cfb	DES-ede-ofb	rc2-ecb
Bf-cfb	DES-ecb	DES-ofb	rc2-ofb
Bf-ecb	DES-ede3-cbc	DESx-cbc	rc4
Bf-ofb	DES-ede3-cfb	Idea-cbc	rc5_ofb
cast5-cbc	DES-ede3-ecb	Idea-cfb	rc5-cbc
cast5-cfb	DES-ede3-ofb	Idea-ecb	rc5-cfb
cast5-ecb	DES-ede-cbc	Idea-ofb	rc5-ecb
cast5-ofb	DES-ede-cfb	rc2-cbc	
DES-cbc	DES-ede-ecb	rc2-cfb	

Where:

DES	Data Encryption Standard
DESx	Variant of DES
Rc4	Rivest Cipher 4
Idea	International Data Encryption Algorithm
Rc2	Rivest Cipher 2
Bf	Blowfish
Cast5	Algorithm designed using the CAST design procedure. AKA Cast-128
Rc5	Rivest Cipher 5
Cbc	Cipher Block Chaining mode
Ecb	Electronic Code Book mode
Ofb	Output Feedback mode
Cfb	Cipher Feedback mode
ede	Encrypt-Decrypt-Encrypt mode
Ede3	Variation of EDE

HTA() Function

Get Hex Value of String

Format HTA(*string*[\$[,**ERR**=*stmtref*])

Where:

string\$ String expression to convert to hexadecimal format.

stmtref Program line number or label to transfer control to.

Returns Hex value of your string.

Description The **HTA()** function returns the hexadecimal value of a given character string. The string returned by the **HTA()** function is always twice the length of the original string and consists solely of the (hex) characters 0-9 and A-F.

Examples

```
0020 LET A$=HTA("CAT") ! yields 434154
0030 LET A$=HTA("01") ! yields 3031
0040 LET A$=HTA(BIN(10,2)) ! yields 000A
```

HWN() Function *Highest Unused Window Number*

Format `HWN(chan [, ERR=stmtref])`

Where:

chan Channel or logical file number of the terminal device whose highest unused window number is to be reported (usually device (0) or the console)

stmtref Program line number or label to transfer control to.

Returns Highest unused window number

Description The **HWN()** function returns an integer, reporting the *highest* unused window number for a given file. If all window numbers are in use, this function will return -1. If the file is not open or not a terminal, ProvideX returns `Error #13: File access mode invalid`.

You can use the window number returned by the function in a subsequent '**WINDOW**' mnemonic to create a new window. If no window number is passed to the '**WINDOW**' mnemonic, it will utilize the *lowest* unused window number.

The maximum number of windows allowed on a terminal device is 250.



Note: The **HWN()** function is affected by your setting for the '**BO**' System Parameter, [p.656](#).

I3E() Function

Convert to/from IEEE Format

- Formats**
1. *Convert From Internal Value to Floating Point:* **I3E(num[,ERR=stmtref])**
 2. *Convert From Floating Point to Internal Value:* **I3E(string\$[,ERR=stmtref])**

Where:

- num** Numeric value to convert to 8-byte IEEE format (scientific notation).
string\$ 8-byte string expression to convert from IEEE format to a numeric value.
stmtref Program line number or label to transfer control to.

Returns Numeric data, IEEE converted to/from ProvideX internal values.

Description The **I3E()** function converts data to/from IEEE floating point format and ProvideX internal values. The primary purpose of this function is to allow for the conversion of data between ProvideX and other applications.

Format 1: Convert From Internal Value to Floating Point

I3E(num[,ERR=stmtref])

If the function is passed a numeric expression, it will return an 8-byte string containing the IEEE floating point value of the number.

Format 2: Convert From Floating Point to Internal Value

I3E(string\$[,ERR=stmtref])

If the function is passed an 8-byte string, the string will be converted to an internal ProvideX numeric value.

Example

```
0010 ! Program to convert values to square roots
0020 OPEN (1,ISZ=8) "FLTDTA"
0030 I=0
0040 LOOP: READ RECORD (1,IND=I,ERR=DONE) F$
0050 F$=I3E(SQR(I3E(F$)))
0060 WRITE RECORD (1,IND=I) F$
0070 LET I=I+1; GOTO LOOP
0090 DONE: CLOSE(1)
0100 END
```


IND() Function

Return Next Record Index

Format **IND**(*chan* [, *fileopt*])

Where:

chan Channel or logical file number of the file to reference.

fileopt Supported file options (see also, [File Options, p.810](#)):

ERR=*stmtref* Error transfer

END=*stmtref* End-Of-File transfer

KEY=*string*\$ Record key

KNO=*num* | *name*\$ File access key number (*num*) or name (*name*\$)

RNO=*num* Record number

stmtref Program line number or label to transfer control to.

Returns Record index (next or specified record).

Description The **IND()** function returns the record index of either the next record in the file specified or, on a Direct or Keyed file using the **KEY=** option, the record index for the record identified by the key.

For Direct/ Keyed or Sort files without the **KEY=** option, the value returned is the record index for the record with the next higher key value.

For variable length Direct/Keyed files, the value returned is an internal pointer to the record.

For more information, see *File Handling* in the *ProvideX User's Guide*.

Example

```
0010 OPEN (13) "CUSTNO"
0020 LET I=IND(13,END=1000)
0030 READ (13,IND=I) R$
0040 PRINT "Rec#: ",I," Data: ", R$
0050 GOTO 0020
1000 PRINT "End-of-file"
1010 END
```

INT() Function

Return Integer Portion

Format `INT(num[,ERR=stmtref])`

Where:

num Numeric expression whose integer portion is to be returned.

stmtref Program line number or label to transfer control to.

Returns Integer portion of numeric value.

Description The `INT()` function returns the integer portion of the value specified. No rounding is performed on the value. The fractional part of the value is truncated.

Examples `A=INT(3.23) ! yields A=3`
`A=INT(-5.6) ! yields A=-5`
`A=INT(.9999) ! yields A=0`

IOL() Function

Get IOList Specification

Formats 1. In Composite String: **IOL(composite\$[,ERR=stmtref])**

2. In Open File: **IOL(chan[,ERR=stmtref])**

Where:

composite\$ Composite string variable whose IOLIST is to be retrieved. String expression.

chan Channel or logical file number whose default IOLIST is to be returned.

stmtref Program line number or label to transfer control to.

Returns Object value of an IOList or composite string variable.

Description The **IOL()** function returns the object code value of an IOList for either an open file or a composite string variable. If the specified file or string variable does not have an associated IOLIST, ProvideX returns Error #81: Invalid IOLIST specification.

Syntax	Returns
IOL(<i>chan</i> :*)	Returns the IOList for the file's embedded data dictionary
IOL(<i>chan</i> :^)	Returns the alternate IOList
IOL(<i>chan</i> :KEY)	Returns the IOList for the file's external key, if any.



Note: To convert the object code into a format you can display, pass it to the **LST()** function.

Examples

```
0100 DIM CUST$:IOL=0110
0110 IOLIST NAME$,ADR1$,ADR2$,SMAN$
0120 PRINT LST(IOL(CUST$))
-:run
IOLIST NAME$,ADR1$,ADR2$,SMAN$
```

```
0100 OPEN (1,IOL=*)"CUSTDB" ! Open with internal IOL
0110 READ DATA FROM "" TO IOL=IOL(1) ! Clears the IOList
```

IOR() Function

Logical OR

Format **IOR**(value1[\$],value2[\$][,ERR=stmtref])

Where:

stmtref Program line number or label to transfer control to.

value1[\$] Compared values. String or numeric expressions/variables. If strings,
value2[\$] *value1\$* must be the same length as *value2\$*

Returns Result of logical 'OR' comparison of two expressions/variables.

Description The **IOR()** function performs a bit-wise logical 'OR' comparison of two string or numeric expressions/variables, and generates a value as a result. The length of the two string expressions must be equal or ProvideX returns an Error #46: Length of string invalid.

<i>Binary</i>	<i>Result</i>
0 IOR 0	0
1 IOR 0	1
0 IOR 1	1
1 IOR 1	1

Therefore

IOR(\$41\$, \$42\$) *yields* Hex 43 01000011

IOR(\$41\$, \$25\$) *yields* Hex 65 01100101

IOR(\$5A\$, \$DD\$) *yields* Hex DF 11011111

See Also **XOR() Exclusive OR Comparison, p.554**
AND() Logical AND, p.394

Example

```
0040 READ (1,END=1000)F$
0050 R$=IOR(F$(1,2),$8080$) ! Turn on high bit
0060 ...
```

JST() Function

Justify String

Format **JST(string\$,len[,jstcode[\$]][,char\$][,ERR=stmtref])**

Where:

char\$ Optional string. Its first character is used to pad *string\$*. If you omit this, the default is to pad with blanks. String expression.

len Desired length of string. Numeric expression.

jstcode[\$] Optional numeric or string parameter defining how to justify the string:

0 *or* **R** Right justify.

1 *or* **L** Left justify - *default*.

2 *or* **C** Centre in string

If omitted, the string is left justified.

stmtref Program line number or label to transfer control to.

string\$ String expression to be processed.

Returns Value used in **OPT=** option.

Description The **JST()** function converts a given character string (*string\$*) to the length (*len*) specified. It makes the string the desired length either by truncating the *string\$* or by appending a defined pad character. The default is to pad with spaces.

If the length you specify is less than zero, ProvideX returns an Error #41: Invalid integer encountered (range error or non-integer).

See Also [PAD\(\) Pad/Truncate String, p.496](#)

Examples The following code sample uses asterisks to justify a numeric value to a length of 30 characters:

```
00180 LET chq_amt=1.98,cust_name$="ACME INC."
00190 LET chq_amt$="*****"+JST(STR(chq_amt),30,"*")
00200 PRINT 'CS',@(0,5),"Customer name :",JST(cust_name$,20)," | ",
00210 PRINT @(0,6),chq_amt$
-:run
Customer name :ACME INC.          |
*****1.98*****
```

This example illustrates the use of alphanumeric versus numeric pad types in the **JST()** function:

```
0100 ! ^100 - PAD and JST functions
0110 Orig$="Test String",Char$=".",PadLen=20
0120 print 'LF',"Original String: "+@(24)+'BR'+Orig$+'ER'+'LF'
0130 Type=0,Type$="L"; gosub JustifyIt; print
0140 Type=1,Type$="R"; gosub JustifyIt; print
0150 Type=2,Type$="C"; gosub JustifyIt
0160 stop
0170 !
0200 ! ^100
0210 JustifyIt:
0220 print "PAD(Orig$, "+str(PadLen)+", "+quo+Type$+quo+", "+quo+Char$+quo,
0230 print ") = "+@(24)+'BR'+pad(Orig$,PadLen,Type$,Char$)+'ER'
0240 print "JST(Orig$, "+str(PadLen)+", "+quo+Type$+quo+", "+quo+Char$+quo,
0250 print ") = "+@(24)+'BR'+jst(Orig$,PadLen,Type$,Char$)+'ER'
0260 return
```

JUL() Function

Return Julian Date

Formats

1. Julian from Numeric: **JUL**(year,month,day[,ERR=stmtref])
2. Julian from Day Format: **JUL**(string\$[,ERR=stmtref])

Where:

- year** Numeric expression of the year. If your value is less than 100, the current century is added to the value.
- month** Numeric expression of the month.
- day** Numeric expression of the day of the month.
- string\$** String in the same format as the **DAY** variable. String expression.
- stmtref** Program line number or label to transfer control to.

Returns

Julian date (converted from year, month, day).

Description

The **JUL()** function is used to convert a date from year, month, day to a Julian date. The Julian date is an integer: the number of days since the system base-date. By default, in ProvideX this is January 1, 1970. Use the **'BY'** system parameter to change the base date.

Historically the true Julian calendar starts sometime around 4713 BC., but because of errors in early calendars, dates prior to 1200 are not reliable. If you want the **JUL()** function to return dates based roughly on the Julian calendar, set the **'BY'** system parameter to 0 zero.



Note: To get the *current* Julian date, use the format `JUL(0,0,0)`.

See Also

[DAY_FORMAT Directive, p.64](#)
[DAY System Variable, p.557](#)
[DTE\(\) Convert Date, p.422](#)
['BY'= System Parameter, p.658.](#)

Examples

The following example converts a given date to Julian format and calculates the difference from the current Julian date:

```
0010 INPUT "Enter Date (MM/DD/YY):",X$:"00/00/00"
0020 LET M=NUM(X$(1,2))
0030 LET D=NUM(X$(3,2))
0040 LET Y=NUM(X$(5,2))
0050 LET N=JUL(Y,M,D,ERR=0100)
0060 PRINT "That is ",N-JUL(0,0,0)," days from now"
0070 STOP
0100 PRINT "Invalid date"; GOTO 0010
```

```
-:RUN
Enter Date (MM/DD/YY):03/31/99
That is 23 days from now
-:
```

In the following example, the **JUL()** function accepts a valid **DAY** string and returns the corresponding Julian date:

```
X$="01/01/A0"
DAY_FORMAT "MM/DD/AA"
PRINT JUL(X$)
  10957
PRINT DTE(10957:"%Y %M %D")
2000 January 1
```

JUL() can be used to determine if a given date is either Saturday or Sunday. Since the **JUL()** function returns a day number, **JUL (...) | 7** would provide a week day number in the range 0-6. Depending on what is configured as the base year (standard PVX is 1970, compatibility mode is 4714 BC) the day number changes; e.g.,

If **'BY'=1970**,

```
Sun=3, Mon=4, Tue=5, Wed=6, Thu=0, Fri=1, Sat=2
```

The weekend can be tested as follows:

```
if ((jul(y,m,d)+3)|7)>=5 then <WEEKEND>
```

If **'BY'=0**,

```
Sun=6, Mon=0, Tue=1, Wed=2, Thu=3, Fri=4, Sat=5
```

The weekend can be tested as follows:

```
if (jul(y,m,d)|7)>=5 then <WEEKEND>
```


KEC() Function

Return Key of Current Record

Format **KEC**(*chan* [, *fileopt*])

Where:

chan Channel or logical file number of your given file.

fileopt Supported file options (see also, [File Options, p.810](#)):

ERR=*stmtref* Error transfer.

KNO=*num* | *name*\$ File access key number (*num*) or name (*name*\$).

stmtref Program line number or label to transfer control to.

Returns Current key or current logical position in file

Description The **KEC()** function returns the current record's key. The result is based on:

- the current file access key or,
- the access key specified using the **KNO=** option.

The current logical position is the last key value specified on a [READ](#) or [EXTRACT](#) regardless of whether a record was found.

Typically, the **KEC()** function is used after you read a record, to determine the key of the record you just read (i.e., the key of the current record or current position in the file). For more information, see *File Handling* in the *ProvideX User's Guide*.

Example

```
0010 OPEN (13) "CUSTNO"
0020 READ (13,KNO=0) R$
0030 LET K$=KEC(13), K1$=KEC(13,KNO=1)
0040 PRINT "Key: ",K$," Alt: ", K1$," Data: ", R$
0050 GOTO 0020
1000 PRINT "End-of-file"
1010 END
```

KEF() Function

Return First Key of File

Format **KEF(*chan*[,*fileopt*])**

Where:

chan Channel or logical file number of the file to reference.

fileopt Supported file options (see also, **File Options**, p.810):

END=stmtref End-of-File transfer

ERR=stmtref Error transfer

KNO=num | name\$ File access key number (*num*) or name (*name\$*)

stmtref Program line number or label to transfer control to.

Returns Key of first record in file.

Description The **KEF()** function returns the key of the first record in the file specified. The result is based on:

- the current file access key or,
- the access key specified using the **KNO=** option.

The **KEF()** function can also be used in TCP (Transmission Control Protocol) to identify local sockets. For more information, see *File Handling* in the *ProvideX User's Guide*.

Example

```

0010 OPEN (1) "CUSTNO"
.....
0100 F$=KEF(1),L$=KEL(1)
0110 PRINT "Customers range from ",F$,"->",L$
.....
1010 END

```

KEL() Function

Return Last Key of File

Format **KEL(*chan*[,*fileopt*])**

Where:

chan Channel or logical file number of the file to reference.

fileopt Supported file options (see also, **File Options**, p.810):

END=stmtref End-of-File transfer

ERR=stmtref Error transfer

KNO=num | name\$ File access key number (*num*) or name (*name\$*).

stmtref Program line number or label to transfer control to.

Returns Key of last record in file.

Description The **KEL()** function returns the key of the last record in the file specified. The result is based on:

- the current file access key or,
- the access key specified using the **KNO=** option.

ProvideX supports the **KEL()** function for ODBC files. For more information, see *File Handling* in the *ProvideX User's Guide*

Examples

```

0010 OPEN (1) "CUSTNO"
.....
0100 F$=KEF(1),L$=KEL(1)
0110 PRINT "Customers range from ",F$,"->",L$
.....
1010 END

```

KEN() Function

Return Key After Next

Format **KEN(chan[,fileopt])**

Where:

chan Channel or logical file number of your given file.

fileopt Supported file options (see also, [File Options, p.810](#)):

END=stmtref End-of-File transfer

ERR=stmtref Error transfer

KNO=num | name\$ File access key number (*num*) or name (*name\$*).

stmtref Program line number or label to transfer control to.

Returns Key of the record that follows the next record in the file.

Description The **KEN()** function returns the key of the record which directly follows the next record in the file specified. The result is based on:

- the current file access key or,
- the access key specified using the **KNO=** option.

For ODBC files, the **KEN()** function supports some debugging tools:

KEN (nn) Returns last generated SQL statement passed to the ODBC driver

KEN (nn, IND=1) Returns cursor name for database

KEN (nn, IND=2) Returns ODBC handles.

For more information, see *File Handling* in the *ProvideX User's Guide*

Example

```

0010 OPEN (13)"INVDET"
0020 K$=KEY(13,END=1000)
0030 REM   Last record is total line - different format
0040 KN$=KEN(13,END=0080); IF KN$(1,6)<>K$(1,6) GOTO 0080
0050 READ (13)IOL=8010 ! Get record
0060 GOSUB 7000 ! Process invoice line
0070 GOTO 0020
0080 READ (13)IOL=8010 ! Get total line
0090 GOSUB 7500; GOTO 0020
1000 END

```

To return the current ODBC SQL statement being passed to the ODBC driver:

```

OPEN (1) "[odb]Database;customer;key=custid"
READ (1,KEY="MIKE")
PRINT KEN(1)
SELECT * FROM customer WHERE custid = 'MIKE'

```

KEP() Function

Return Prior Record's Key

Format **KEP(chan[,fileopt])**

Where:

chan Channel or logical file number of your given file.

fileopt Supported file options (see also, **File Options**, p.810):

END=stmtref End-of-File transfer

ERR=stmtref Error transfer

KNO=num | name\$ File access key number (*num*) or name (*name\$*).

stmtref Program line number or label to transfer control to.

Returns Key of the prior record in file

Description The **KEP()** function returns the key of the record prior to the record in the file specified. The result is based on:

- the current file access key or,
- the access key specified using the **KNO=** option.

If the current record is at the start of the file, there is no prior record. ProvideX reports Error #2: END-OF-FILE on read or File full on write unless you include an **ERR=** or **END=** option.

ProvideX supports the **KEP()** function for ODBC files. For more information, see *File Handling* in the *ProvideX User's Guide*.

Example

```

0010 OPEN (13)"CUSTNO"
0020 READ (13,KEY="zzz",DOM=30) ! Go to End of file
0030 LET K$=KEP(13,END=1000)
0040 READ (13,KEY=K$)R$
0050 PRINT "Key: ",K$," Data: ",R$
0060 GOTO 0030
1000 PRINT "Back to the start"
1010 END

```

KEY() Function

Return Key of Next Record

Format **KEY**(*chan* [, *fileopt*])

Where:

chan Channel or logical file number of the file to reference.

fileopt Supported file options (see also, [File Options, p.810](#)):

END=*stmtref* End-of-File transfer

ERR=*stmtref* Error transfer

KNO=*num* | *name*\$ File access key number (*num*) or name (*name*\$)

IND=*num* Record index

RNO=*num* Record number.

stmtref Program line number or label to transfer control to.

Returns Key of next record in file

Description The **KEY()** function returns the key of either the next record in the file specified or, via the **IND=** or **RNO=** options, the key of the record at the index/record number specified. The result is based on:

- the current file access key or,
- the access key specified using the **KNO=** option.

For more information, see *File Handling* in the *ProvideX User's Guide*.

Example

```
0010 OPEN (13)"CUSTNO"
0020 LET K$=KEY(13,END=1000)
0030 READ (13,KEY=K$)R$
0040 PRINT "Key: ",K$," Data: ",R$
0050 GOTO 0020
1000 PRINT "End-of-file"
1010 END
```

KGN() Function

Generate Record Key

Format **KGN**([*ext_key*\$],[*data*\$],[*key_def*\$],[*key_num*],[**ERR**=*stmtref*])

Where:

ext_key\$ Value of the external key. Optional. String expression.

data\$ Contents of the data record. String expression.

key_def\$ Key definition structure. String expression. This can be extracted using the **FIB()** function at position 85 (in native ProvideX mode) or using the **FIN()** function at position 86 for a length of 385 (in BBx emulation mode.)

key_num Key number (primary or an alternate key) to extract. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Key of record in file, given the record's contents.

Description The **KGN()** function returns a string comprising the key of the record provided. This function can be used to determine the value of an alternate (or primary) key of a record, given the record's contents (and external key, if present).

When comparing keys with descending segments, an application can specify the key number as a negative value. In this case, the descending segments will be inverted so that a logical compare will function properly.

Example

```
0010 OPEN (13)"Cstfile"
0020 IF PRM('BX')=0 THEN X$=MID(FIB(13),85) ELSE X$=MID(FIN(13),86,385)
0030 K$=KEY(13,END=0070)
0040 READ RECORD (13,KEY=K$)R$
0050 PRINT "Key: ",K$," Alt: ",KGN(K$,R$,X$,1)
0060 GOTO 0030
0070 PRINT "End-of-file"
0080 END
```

LCS() Function

Return Lowercase String

Format `LCS(string$[,ERR=stmtref])`

Where:

string\$ String expression whose lower case ASCII counterpart is to be returned.

stmtref Program line number or label to transfer control to.

Returns Lowercase counterpart of string

Description The `LCS()` function returns the lower case counterpart of the original string (with all upper case alphabetic characters replaced by their corresponding lower case characters).

See Also [UCS\(\) Return Upper Case String, p.546](#)
[DEF systab= Directives, p.74.](#)

Example

```
0010 INPUT "Enter name: ",NAME$
0020 LET NAME$(2)=LCS(NAME$(2))
0030 LET NAME$(1,1)=UCS(NAME$(1,1))
0040 PRINT "Name is",@(10),": ",NAME$
-:RUN
Enter name: SMITH
Name is   : Smith
```



Note: If you use an *asterisk* * instead of a string (i.e., `LCS(*)`) the function returns the 256 byte Lowercase Conversion Table.

LEN() Function

Return String Length

Format `LEN(string$[,ERR=stmtref])`

Where:

string\$ String expression whose length is to be returned.

stmtref Program line number or label to transfer control to.

Returns Integer, length of given string, 0 *zero* if null string.

Description The **LEN()** function returns an integer reporting the length of the given string. If the given string is a null string ("") then the function returns a length of 0 *zero*.

Examples `A=LEN("HELLO") ! yields 5`
 `A=LEN("") ! yields 0`
 `A=LEN("A"+"BC") ! yields 3`
 `A=LEN(DAY) ! yields 8 (DAY variable is in format MM/DD/YY)`

LNO() Function

Return Line Number

Format `LNO(num[,ERR=stmtref])`

Where:

num Numeric value or variable representing the line number.

stmtref Program line number or label to transfer control to.

Returns Integer, line number of the line specified.

Description The **LNO()** function returns the line number of the line specified in its argument. This can be used in an error handler to check where an error has occurred; e.g.,

```
IF TCB(5)>LNO(Client_upd) THEN ...
```

This can also help to resolve problems where line numbers are hard-coded in a program and then the **RENUMBER** directive is used to change them.

Examples

`LNO(1000) ! returns 1000`

`LNO(Label1) ! returns the line number where Label1 is.`

LOG() Function

Return Base 10 Logarithm

Format `LOG(num[,ERR=stmtref])`

Where:

num Numeric expression whose base ten logarithm is to be returned.

stmtref Program line number or label to transfer control to.

Returns Numeric, base ten logarithm.

Description The **LOG()** function returns the numeric base ten logarithm of a given number, rounded to the current **PRECISION** setting. This is the inverse of the **EXP()** function. ProvideX returns Error #40: Divide check or numeric overflow if the numeric value is negative.

Examples `0010 PRINT EXP(LOG(90)/3) ! Print cube root of 90`

LRC() Function Longitudinal-Redundancy Check

Format `LRC(string$[,ERR=stmtref])`

Where:

string\$ Character string whose longitudinal redundancy checksum is to be calculated.

stmtref Program line number or label to transfer control to.

Returns One-byte string, longitudinal checksum.



Note: The **LRC()** function is used primarily in conjunction with synchronous communications.

Description The **LRC()** function returns the longitudinal redundancy checksum of a character string. The longitudinal redundancy check of a character string is a one byte string resulting from a logical **XOR()** comparison of all the characters in the string.

Examples

```
A$=HTA(LRC($0102$)) ! yields hex 03
A$=HTA(LRC($0305$)) ! yields hex 06
```

LST() Function Return List Form of Statement

Formats **LST([EDIT][*]*internal*\$[,ERR=*stmtref*])**

Where:

- *** *Asterisk*. Returns the listing in colourized syntax.
- EDIT** Keyword indicating that listing is to be returned with indented format.
- internal*\$** Character string containing the internal (compiled) form of a ProvideX statement. String expression; e.g., X\$=LST(PGM(10)).
- stmtref*** Program line number or label to transfer control to.

Returns List format from compiled statement.

Description The **LST()** function converts a ProvideX statement from internal form to normal source format. You must ensure that the string processed by the **LST()** function is a statement in valid internal form. If it is not valid, ProvideX returns either **Error #30: Statement too complex -- cannot compile** or **Error #49: <*> Internal program format error <*>**.

Use the **EDIT** keyword to return a formatted statement and the *asterisk* ***** to display colourized syntax (the **'CS'** parameter must be on); e.g.,
X\$=LST(EDIT *PGM(10)).



Note: There is a *CMD command line utility called COLOUR (or COLOR) that can be used to display or alter the current settings. Typing COLOUR or COLOR at a ProvideX prompt will display online help for this utility.

See Also [*'H' Mnemonic, p.613](#)
['CS' System Parameter, p.660](#)

Examples

```
0030 INPUT "Enter statement to display:",A
0035 IF A=0 THEN GOTO 2000
0040 LET X$=PGM(A,ERR=1000)
0050 PRINT LST(X$)
0060 GOTO 0030
1000 PRINT "Cannot find statement"
1010 GOTO 0030
2000 PRINT "DONE"; STOP
-:run
Enter statement to display:50
0050 PRINT LST(X$)
Enter statement to display:
DONE
```

MAX() Function

Return Maximum Value

Format **MAX** (*compare1,compare2,...[,ERR=stmtref]*)

Where:

compare1, compare2, ... Comma-separated list of numeric values and/or expressions to be compared.

stmtref Program line number or label to transfer control to.

Returns Numeric, largest value, given list of numerics.

Description The **MAX()** function evaluates and returns the maximum (largest) value of the numeric values or expressions specified. There is no limit to the number of values or expressions that can be passed to the **MAX()** function.



Note: You can also use the ProvideX **MAX()** function to obtain statistical information on ODBC data fields.

See Also [MIN\(\) Return Minimum Value, p.481](#)

Examples In this example, **MAX()** evaluates an expression ($12 * 3.7 = 44.4$), a literal (44.8) and a variable (evaluated, $A = 43.2$):

```
0050 LET A=13.21*3.27
0060 LET B=MAX(12*3.7,44.8,A)
0070 PRINT B
->run
44.8
```

MEM() Function

Return Memory Value

Formats

1. *Get Address of String*: **MEM**(var\$[,ERR=stmtref])
2. *Read Memory, 2 Bytes*: **MEM**(address[,ERR=stmtref])
3. *Read Memory, 'n' Bytes*: **MEM**(address,bytes[,ERR=stmtref])
4. *Change Memory*: **MEM**(address,val\$[,ERR=stmtref])

Where:

- var\$* Name of string variable whose address you wish to obtain.
- address* Memory address being referenced. Numeric expression.
- bytes* Number of bytes to return. Numeric expression.
- val\$* Value you wish to write. String expression.
- stmtref* Program line number or label to transfer control to.

Returns

Memory location and information.



Note: This function is mainly for use with external functions; i.e., using the functions **DLL() Call Windows DLL**, p.418.

Description

The **MEM()** function provides direct access to memory location through the use of pointers. ProvideX performs address validation and returns Error #41: Invalid integer encountered (range error or non-integer) for an invalid memory location.

Format 1 returns the address of your string variable.

Format 2 returns the contents of a word (16 bits) of memory. The value returned is a binary value (integer in two's complement format).

Format 3 returns a string consisting of the data at the address specified for the specified number of bytes.

Format 4 copies your character string value to the address given.

MID() Function

Return Substring

Format **MID(string\$,offset[,len][,ERR=stmtref])**

Where:

len Length of the substring.

offset Starting position of the substring. Numeric expression, integer. If the integer is negative, the offset is taken from the end of the string.

stmtref Program line number or label to transfer control to.

string\$ String expression whose hash value is to be returned.

Returns Extracted portion of string (similar to substring).

Description Use the **MID()** function to extract a portion of a string. Using this function is similar to using a substring, except that it can be used directly with the return value of a function, variable or expression; e.g.,

```
->IF MID(MSE,22,1)>$00$ AND MID(MSE,22,1)<$FF$ THEN %WDX$="[WDX]"
```

In addition, if the offset is negative, ProvideX uses it as an offset from the end of the string. For example, `MID(X$, -1)` is the last character of X\$. If the length is negative, then ProvideX uses it as the number of characters preceding the offset. That is, `MID("ABCD", -1, -1)` returns C (the first character preceding the last character) and `MID("abcde", -2, -4)` yields abc.

By default, if this function is passed an invalid offset, it returns a null string. If passed an invalid length, then it returns the rest of the string.

Example `F_KSZ=DEC(00+MID(FID(0),11,1))`

MIN() Function

Return Minimum Value

Format `MIN(compare_1,compare_2, ... [,ERR=stmtref])`

Where:

compare1, compare2, ... Comma-separated list of numeric values and/or expressions to be compared.

stmtref Program line number or label to transfer control to.

Returns Numeric, smallest value, given list of numerics.

Description The **MIN()** function returns the minimum (smallest) value of the numeric values or expressions specified. There is no limit to the number of values or expressions that can be passed to the **MIN()** function.



Note: You can also use the **MIN()** function to obtain statistical information on ODBC data fields.

See Also [MAX\(\) Return Maximum Value, p.478](#)

Example In this example, the **MIN()** function evaluates an expression ($13/2.96=4.39$), a literal (4.5) and a variable (evaluated, $A=5.69$):

```
0010 A=12.345/2.71
0020 B=MIN(13/2.96,4.5,A); PRINT B
-:run
4.39
```

MNM() Function

Return Mnemonic Value

Format `MNM(mnemonic[$[,chan]][,ERR=stmtref])`

Where:

chan Channel or logical file number of the given file. If omitted, the default is file 0.

mnemonic\$ Name of defined mnemonic to look up, or the string value of the mnemonic. String expression.

stmtref Program line number or label to transfer control to.

Returns String, command sequence for mnemonic.

Description The **MNM()** function returns the defined command sequence for the specified mnemonic on the file given. The command sequence is typically the exact transmission string to handle the mnemonic for the file.

The mnemonic must have been predefined using the **MNEMONIC** directive. The **MNM()** function returns a null string if the mnemonic has not been defined.

See Also [MNEMONIC Directive, p.210](#)
[Chapter 5. Mnemonics, p.577.](#)

Example See if the terminal supports condensed print. If so, create window:

```
IF MNM('CP') = " "
  THEN PRINT 'WINDOW' (0,0,80,25), 'CS',
  ELSE PRINT 'WINDOW' (0,0,132,30), 'CP', 'CS',
```

MOD() Function

Return Modulus

Formats **MOD(num,base[,ERR=stmtref])**

Where:

base Modulus base value. Numeric expression.

num Value for which to calculate the modulus. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Numeric, modulus / remainder.

Description The **MOD()** function returns the modulus / remainder from a division of the first expression by the second.



Note: You can also use the pipe | operator to return a modulus value. The syntax is as follows: *num|base,ERR=stmtref*.

Examples

A=MOD(10,3) ! yields A=1

A=MOD(10,5) ! yields A=0

A=MOD(9,3.5) ! yields A=2

The following conditions deal with leap year dates:

```
1040 IF MOD(Y,4)=0 THEN LET N.MAX=366,M.TBL$(3,2)="29" ELSE LET N.MAX=365,
1040:M.TBL$(3,2)="28"
```

Use of the pipe "|" is the equivalent of the syntax in line 1040 above:

```
1040 IF Y|4=0 THEN LET N.MAX=366,M.TBL$(3,2)="29" ELSE LET N.MAX=365,
1040:M.TBL$(3,2)="28"
```

MSG() Function

Return Message Text

Formats

1. *By Message Number*: **MSG(err_msg,[ERR=stmtref])**
2. *By Message Key*: **MSG(msg_key\$,[param1, param2, ...][,ERR=stmtref])**
3. *Message Library Setting*: **MSG(*)**

Where:

- * **MSG(*)** returns the current **MESSAGE_LIB** settings. If more than one **MESSAGE_LIB** is open, the function returns all the open filenames in search order, each separated by the default field separator, **SEP** (e.g., \$8A\$).
- err_msg** Number of the error message to return. Numeric expression. If **err_msg** is a positive integer, it returns the associated message, as described under **Error Codes and Messages, p.828**. If **err_msg** is -1, it returns extended or external error information.
- msg_key\$** Message key to the Message Library file. String expression.
- param1, param2, ...** Optional parameters. You can use a list of values to replace the parameters stored in the message.
- stmtref** Program line number or label to transfer control to.

Returns

Text associated with given message number or key.

Description

The **MSG()** function returns the text of the message whose number or key is specified. Use this function to obtain more information about errors generated in ProvideX by a program and to return information from your own message libraries.

Use the **DEF MSG** directive to temporarily override the **MSG()** function.

See Also

[DEF MSG Directive, p.70](#)
[MESSAGE_LIB Directive, p.208](#)
[ERR System Variable, p.560.](#)
[Error Codes and Messages, p.828](#)

Examples

The following examples illustrate the different uses for the **MSG()** function.

Example 1:

```
0010 OPEN (1,ERR=1000)"PRINTR"
0020 OPEN (2,ERR=1000)"CUSTOM"
0030 READ (2,KEY=" ",ERR=1000)R$
0040 ...
1000 PRINT "Could not open PRINTR",'LF',MSG(RET)
1010 STOP
```

Example 2:

```
KEYED "MESSAGE.LIB",20,0,-256
OPEN (1)"MESSAGE.LIB"
WRITE RECORD (1,KEY="NOCUST")"Sorry but Customer %1 is not valid"
CLOSE (1)
MESSAGE_LIB "MESSAGE.LIB"
PRINT MSG("NOCUST","0001")
Sorry but Customer 0001 is not valid
```

Example 3:

```
0010 ! Returns the current message library name, if any is in effect
0020 PRINT MSG(*)
```

Example 4:

Use this function to obtain the ProvideX error message associated with an error number:

```
-: ?msg(14)
Error #14: Invalid I/O request for file state
```

MSK() Function

Scan String for Mask

Format MSK(*string*,\$*mask*\$[,ERR=*stmtref*])

Where:

mask\$ String containing the pattern / mask definition. If this value is null, then the previously used pattern is reused. String expression.

stmtref Program line number or label to transfer control to.

string\$ String to search. Maximum string size 8kb.

Returns Integer reporting starting offset.

Description Use the **MSK()** function to scan a string looking for a specific pattern of characters. The value returned is an integer reporting the starting offset of the longest string matching the given mask or pattern. The pattern defines the mask as a regular expression or series of characters, some of which have a special meaning. The following table lists those characters and their meanings. Combinations are allowed.

Mask Character.	Format in Pattern\$	Search
^ <i>Caret</i>	At start of regular expression.	To find a match with the start of the string being searched.
\$ <i>Dollar Sign</i> \$	At end of regular expression.	To find a match with the end of the string being searched.
. <i>Period</i>		To find a match with any character
(<i>string</i>)	String of characters (or other codes) enclosed in parentheses.	To define a sub-expression to match.
[<i>string</i>]	String enclosed in square brackets	To find a match with any character in that string.
[^<i>string</i>]	Square bracketing combined with a caret ^ as the first character of the string.	To find a match with any character except the characters in the string; e.g., [xyz] matches x, y, or z, while [^xyz] matches a, b, c, but not x, y, or z.
- <i>Dash</i>	2 characters separated by - <i>dash</i> .	To specify a range.
[<i>str-ing</i>]	Combination of dash within string in square brackets.	To form expressions: e.g., [a-bd-z] to search for a match with any lower case letter except c.

Mask Character.	Format in Pattern\$	Search
* Asterisk	At the end of a character (or sub-expression).	To search for zero or more occurrences of the character (or sub-expression); e.g., in <code>f o*</code> , the <code>*</code> operates on the <code>o</code> ; it matches <code>f</code> , <code>f o</code> , <code>f o o</code> etc.. but doesn't match <code>f a</code> . The expression <code>f (at) *</code> matches <code>f</code> , <code>fat</code> , <code>fatat</code> , <code>fatatat</code> , etc.
+ Plus Sign	At the end of a character (or sub-expression).	To find a match with 1 or more occurrences of that character (or sub-expression); e.g., <code>f a+</code> matches <code>f a</code> , <code>f a a</code> , <code>f a a a</code> , etc. but not <code>f</code> .
? Question Mark	At the end of a character (or sub-expression).	To indicate that it is optional. For example, <code>colou?r</code> matches <code>color</code> or <code>colour</code> and <code>sea(horse)?</code> matches <code>sea</code> or <code>seahorse</code> .
Vertical Bar	Separating two characters (or sub-expressions).	To find a match for either of the two characters (or sub-expressions); e.g., <code>c(at) (ow)</code> matches <code>cat</code> or <code>cow</code> and <code>at (nd)</code> matches <code>at</code> or <code>and</code> .
\ Backslash	Preceding a mask character.	To indicate that the character that follows is to be taken literally; e.g., to search for multiple asterisks use <code>**</code> .

The **MSK()** function returns the starting offset in the search *string*\$ (where it matches the pattern specified). The **MSL** system variable and **TCB(16)** return the length of the string found.



Note: When the **'TL'** (Thoroughbred **LIKE**) system parameter is **OFF**, the **LIKE** operator uses the same pattern matching that **MSK()** does. With the **'OM'** (Old Mask) System Parameter **ON**, **MSK()** behaviour is compatible with the UNIX **GREP** command.

See Also

[MSL System Variable, p.567](#)
[TCB\(\) Return Task Information, p.534](#)
['TL' System Parameter, p.690](#),
['OM' System Parameter, p.678](#).
[Operators, p.821](#)

Example

```
->PRINT MSK("my name is Foxy", "[A-Z][a-z]*"),MSL
12 5
```

Here, **MSK()** reports the starting offset. (F, the 12th character, is the first uppercase character.) **MSL()** reports the length of the string. (Foxy is 5 characters long.)

MXC() / MXL() Functions *Return Maximum Column/Line*

Formats 1. *Return Maximum Columns:* **MXC**(chan[,ERR=stmtref])

2. *Return Maximum Lines:* **MXL**(chan[,ERR=stmtref])

Where:

chan Channel or logical file number of the file to reference, typically 0 (zero, for the *current display window*). Use an integer for a device channel (e.g., a printer). Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Integer, zero-based, maximum columns/lines allowed for file/device.

Description The **MXC()** function returns an integer reporting the zero-based maximum number of columns allowed for a file or device. **MXL()** returns an integer reporting the zero-based maximum number of lines allowed for your given file or device. Use these functions as a quick means to determine the size of the *current display window on a screen*.

The functions **MXC()** and **MXL()** return the maximum available column and line values for the channel based on the current default settings for paper size, printable area, offset, margin, font and pitch.

Examples On a terminal where **MXC(0)=79** and **MXL(0)=24** with **LEN(X\$)=15**:

```
0100 X$="THIS IS A TITLE"
0110 PRINT @(MXC(0)-LEN(X$)+1,0),X$ ! Right-justified on line 1
0200 PRINT @(0,MXL(0)),"F1-Help F4-Quit", ! Left-justified on bottom line
```

The next example returns the maximum column and line values for a given printer (ASIS is the last printer opened on channel 30). The values are zero-based; i.e., the **MXC()** value returned is 79 for 0-79 = 80 columns:

```
OPEN INPUT (30)"*WINPRT*;ASIS"
C=MXC(30)+1 ! For this printer MXC(30) returns 79, C=80 (0-79)
L=MXL(30)+1 ! For this printer MXL(30) returns 55, L=56 (0-55)
CLOSE (30)
```


NEW() Function

Create New Object

Formats

1. Create Object: **NEW(class\$[,param1[\$], param2[\$],...][,ERR=stmtref])**
2. Clone Object: **NEW(obj_id[,ERR=stmtref])**

Where:

<i>class\$</i>	Name of class for creating new object. String expression.
<i>param1[\$], ...</i>	Optional parameters to be entered in the object's ON_CREATE logic.
<i>obj_id</i>	Object identifier of an existing Object. The cloning process creates a duplicate with all properties copied. However, it does not run the On_Create for the new clone nor provide access to files from the original
<i>stmtref</i>	Program line number or label to transfer control to.

Returns

Object Identifier.

Description

The **NEW()** function is used in *Object Oriented Programming* to create a new object based on a specified class name or an existing object (*obj_id*). If the class name already exists, then its definition is used. If it has not been defined previously, the system attempts to load the program *class.pvc* and execute/define the **DEF CLASS** within it (the **DEF CLASS** clause must be at the start of the program). If the system is unable to properly determine the class definition, an Error #90:"Unable to locate Object class definition" is generated.

In the example below, if the class definition for Customer does not already exist in memory, then the system attempts to load the program *Customer.pvc*:

```
Cst = NEW ("Customer")
```

If this is successful, the **NEW()** function returns the object identifier assigned to the object. The label ON_CREATE is called to initialize the object (if ON_CREATE logic exists in the class definition). Optional parameters can also be used with the **NEW()** function to be passed on to the ON_CREATE entry point; e.g.,

```
C = NEW("Customer", File_number)
```

In *Customer.pvx*:

```
0010 ON_CREATE: ENTER File_no
```



Note: Use **REF()** to increment the reference count.

See Also

[DEF CLASS Directive, p.65](#)
[REF\(\) Function, p.512](#)
[Data Integration, User's Guide](#)

NOT() Function Invert String Bits/Logical Condition

- Formats**
1. *Invert String*: **NOT**(data\$[,ERR=stmtref])
 2. *Invert Logical Condition*: **NOT**(num[,ERR=stmtref])

Where:

data\$ Data whose bits are to be inverted. String expression.

num Value for inverting a logical condition. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Inverted value, string or numeric.

Description This function returns the inverted value of a string or numeric (i.e., with all ON bits turned OFF and vice versa).

Format 1: *Invert String*

NOT(data\$[,ERR=stmtref])

The **NOT**() function inverts the value of the bits in the character string specified. The string returned by the **NOT**() function will be the same length as the given data string. All OFF bits in the string will be returned ON, while all ON bits will be returned OFF. The string expression can be a literal like "abc" or \$A\$, a variable such as A\$, or an expression using operators, such as A\$="1234"; e.g.,

```
A$=NOT($0010$)   yields   A$=$FFEF$
A$=NOT($FF00FF$) yields   A$=$00FF00$
A$=NOT($5A5A$)   yields   A$=$A5A5$
```

Format 2: *Invert Logical Condition*

NOT(num[,ERR=stmtref])

Use this format to invert a logical condition. If the result of the condition is 0 (*false*), this function returns 1 (*true*). Otherwise this function returns 0 (*false*); e.g.,

```
IF NOT(CST_ID$=CMPR_ID$)
  THEN GOSUB NO_MATCH
```

NUL() Function

Return Test for Null

Formats **NUL(string\$[,ERR=stmtref])**

Where:

string\$ String expression to be tested for null value.

stmtref Program line number or label to transfer control to.

Returns Numeric true/false code, 1 or 0.

Description The **NUL()** function returns a numeric status code. The value returned is 1 (*true*) if the string expression is null or contains nothing but spaces. If the string contains data other than spaces, then this function returns 0 (*false*).

Example In this example, the **NUL()** conditions force the user back to the prompts to put data in mandatory fields:

```

0010 BEGIN
0020 INPUT EDIT "Enter your Name: ",N$
0030 IF NUL(N$) THEN PRINT "Your name is required. "; GOTO 0020
0040 INPUT EDIT "Address: ",A$
0050 IF NOT(NUL(A$)) THEN GOTO 1000 ELSE GOTO 0040
1000 PRINT "DONE"; STOP
-:RUN
Enter your Name:
Your name is required. Enter your Name: IDA WANNA
Address:
Address:
Address: 123 ANY ST.
DONE
-:?nul(a$),nul(n$)
0 0

```

NUM() Function

Convert String to Value

Formats **NUM**(string\$[,ERR=stmtref])

Where:

string\$ Character string whose value is to be converted to a numeric value.
 Numeric in string expression (e.g., "19990317").

stmtref Program line number or label to transfer control to.

Returns Numeric value from string.

Description The **NUM()** function returns the numeric value of a numeric expression in a string (e.g., 19990317 from "19990317"). The string is evaluated and converted to a numeric value.

If your given string does not contain a valid numeric value, ProvideX returns an Error #26: Variable type invalid. Your string expression can include any number of the following valid characters: 0-9, a space, a comma, an equals sign, or a dollar sign. You can also include one decimal point along with one sign character (either '-' or '+').

NUM() always ignores the decimal point and thousands separator settings in the 'DP' and 'TH' system parameters.

Examples

A=NUM("1.34") ! Yields 1.34

A=NUM("-1,005.") ! Yields -1005

A=NUM("A",ERR=50) ! On error, transfers to 0050 and sets ERR=26

Characters	Information Returned by OBJ()
3, 2	<p>State:</p> <ul style="list-style-type: none"> \$0000\$ Additive bit state: nothing listed below is happening \$0001\$ Drop box open \$0002\$ Control changed: issue CTL on closure \$0004\$ Control has focus \$0008\$ Control is dropped \$0010\$ Insert mode for typing \$0020\$ Temporary font in effect \$0040\$ Ignore format \$0080\$ Temporarily non-paint \$0100\$ Notification pending "signal" button \$0200\$ Focus changed due to signal \$0400\$ Hide until input \$0800\$ Button is down \$1000\$ Focus interrupt is deferred \$2000\$ Input must have implied decimal point \$8000\$ Disabled object <p>For example, to detect enabled / disabled NOMADS control: X\$=OBJ(OBJECT_NAME.CTL) IF AND(X\$(3,2),\$8000\$)=\$8000\$ THEN PRINT "Object is disabled"</p>
5, 2	Column
7, 2	Line
9, 2	Width
11, 2	Height
13, 4	Visual screen attributes at time of creation
17, 4	Operating system handle
21, 2	Left Pixel }... Includes Border
23, 2	Top Pixel }... Includes Border
25, 2	Right Pixel }... Includes Border
27, 2	Bottom Pixel }... Includes Border
29, 2	Left Viewable Column
31, 2	Top Viewable Row
33, 2	Viewable # of Columns
35, 2	Viewable # of Rows

OPT() Function

Return File Open Options

Format **OPT(chan[,ERR=stmtref])**

Where:

chan Channel or logical file number of the file to reference.

stmtref Program line number or label to transfer control to.

Returns Value used in **OPT=** option.

Description The **OPT()** function returns the value used in the **OPT=** option of the **OPEN** directive for the given file. If the file referred to in the **OPT()** function is not open, ProvideX returns an Error #13: File access mode invalid. If you omitted the **OPT=** option for an open file, the **OPT()** function returns a null string.

Use this function primarily with device drivers to deal with such processes as generating banners and multiple copies.

See Also [OPEN Directive, p.232.](#)

Example

```
00040 ! Report OPT( ) value for open COM port:
00050 LET settings$="9600,n,8,1,x"
00060 LET port$="COM2"
00070 OPEN (1,ISZ=1,OPT=settings$)port$
00080 PRINT OPT(1)
00090 END
-:run
9600,n,8,1,x
```

PAD() Function

Pad/Truncate String

Format	PAD(string\$,len[,pad_code][,char\$][,ERR=stmtref])
	<i>Where:</i>
<i>char\$</i>	Optional string. Its first character is used to pad <i>string\$</i> . If you omit this, the default is to pad with blanks. String expression.
<i>len</i>	Desired length of string. Numeric expression.
<i>pad_code</i> [<i>\$</i>]	Optional parameter defining how to pad the string, either numeric or string: 0 or L Pad on Left (right justify) 1 or R Pad on Right (left justify) - <i>default</i> 2 or C Centre in string If omitted, the string is padded to the right.
<i>stmtref</i>	Program line number or label to transfer control to.
<i>string\$</i>	String expression to be processed.

Returns Value used in **OPT=** option.

Description The **PAD()** function converts a given character string (*string\$*) to the length (*len*) specified. It makes the string the desired length either by truncating the *string\$* or by appending the defined pad character. The default is to pad with spaces.

If the length you specify is less than zero, ProvideX returns an Error #41: Invalid integer encountered (range error or non-integer).

See Also [JST\(\) Justify String, p.461](#)

Examples The following code uses asterisks to pad a numeric value to a length of 30 characters:

```
00180 LET chq_amt=1.98,cust_name$="ACME INC."
00190 LET chq_amt$="*****"+PAD(STR(chq_amt),30,"*")
00200 PRINT 'CS',@(0,5),"Customer name :",PAD(cust_name$,20)," | ",
00210 PRINT @(0,6),chq_amt$
-:run
Customer name :ACME INC.          |
*****1.98*****
```

This code sample illustrates the use of alphanumeric versus numeric pad types in the **PAD()** function:

```
0100 ! ^100 - PAD function
0110 Orig$="Test String",Char$=".",PadLen=20
0120 print 'LF',"Original String: "+@(24)+'BR'+Orig$+'ER'+'LF'
0130 Type=0,Type$="L"; gosub PadIt; print
0140 Type=1,Type$="R"; gosub PadIt; print
```



```
0150 Type=2,Type$="C"; gosub PadIt
0160 stop
0170 !
0180 PadIt:
0190 print
      "PAD(Orig$, "+str(PadLen)+", "+pad(str(Type),3,2)+", "+quo+Char$+quo,
0200 print " ) = "+@(24)+'BR'+pad(Orig$,PadLen,Type,Char$)+'ER'
0210 print "PAD(Orig$, "+str(PadLen)+", "+quo+Type$+quo+", "+quo+Char$+quo,
0220 print " ) = "+@(24)+'BR'+pad(Orig$,PadLen,Type$,Char$)+'ER'
0230 return
```

PCK() Function

Pack Numeric Data

Format PCK(*num*[,*size*[,ERR=*stmtref*]])

Where:

num Numeric value to be packed.

size Integer from 1 to 8 specifying the number of characters in the resultant string. If the value is too large for the size of the packed result, excess digits on the left are discarded. If no size is included, then the default is 8.

stmtref Program line number or label to transfer control to.

Returns String expression whose value represents a packed number.

Description PCK() is used to pack a numeric value into a string expression. It is the counterpart of the **UPK()** function.

The packing algorithm used takes a numeric value and splits it into a series of two digit values where each of the two digit values represents a number between 0 and 99. These numbers are then added to 32 to create the series of single-byte printable characters that comprise the packed string. To unpack the value each byte of the string has 32 subtracted from it and the resultant values become a series of 2-digit values in the final result.

Should the value of any two-digit pair (when added to 32) equal or exceed the standard file separator (\$8A\$), the value will be incremented by one when the output string is created. When unpacking the string, any byte exceeding the field separator will be reduced by one prior to subtracting 32.



Note: This function is not necessarily compatible with all Business Basics

See Also **UPK() Unpack Numeric Data**, p.548,
CMP() Compress Data, p.404

PFX() Function

Return Prefix Value

Formats

1. Return Specific Prefix: **PFX(num)[,ERR=stmtref]**
2. Return Prefix for PGN: **PFX(PGN[,ERR=stmtref])**

Where:

num **PREFIX** number to use. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns

String, current value of given **PREFIX**.

Description

The **PFX()** function returns the current value of your given prefix number. If no prefix has been defined, the function will return a null string. If your specific prefix has been disabled, ProvideX returns an Error #14: Invalid I/O request for file state. (Use an **ERR=** option to transfer control on the error.)

You can obtain the **PREFIX FILE** value by using **PFX(-1)**. The **PFX** system variable contains the current **PREFIX(0)** value.

See Also

[PREFIX Directive, p.249](#),
[PFX System Variable, p.568](#)

Example

```

4000 REM
4010 LET I=-1
4020 LET X$=PFX(I,ERR=4040)
4030 PRINT "PREFIX("+STR(I)+")=",X$
4040 IF I<=8 THEN LET I=I+1; GOTO 4020
4050 PRINT "PREFIX PGN=",PFX(PGN);STOP
->GOTO 4000
->RUN

PREFIX(-1)=
PREFIX(0)=\OTHER\===\ \OTHER\CUST\CASHRCPT\ \OTHER\CUST\MSC\ \OTHER\CUST\SALES\
PREFIX(2)=\MANUALS\===\ \PVX\TESTS\
PREFIX(3)=
PREFIX(4)=
PREFIX(5)=
PREFIX(6)=
PREFIX(7)=
PREFIX(8)=
PREFIX(9)=
PREFIX PGN=\OTHER\PGMS\===\ \OTHER\PGMS\PVX\ \OTHER\PGMS\CUST\

```

PGM() Function

Return Program Line

Formats **PGM(*lineno*[,*prog_level*][,*ERR=stmtref*])**

Where:

lineno Statement line number. Numeric expression. Use an integer from 1 to 64999. -1, -2, or -3 returns program names instead; see description below.

prog_level Optional numeric value indicating which program level to return.

stmtref Program line number or label to transfer control to.

Returns String, compiled format of statement.

Description The **PGM()** function returns a string containing the internal (compiled) format of a given program statement number. If the line number is -1, the main (level 1) program name is returned; -2 returns the current program name and -3 returns the complete name of the program as specified on the **CALL/LOAD/PERFORM** or **RUN**.

If the statement number does not exist, and an **ERR=** option is specified, ProvideX returns Error #21: Statement number is invalid and transfers control to *stmtref*. If the statement number does not exist, and the **ERR=** option is omitted, ProvideX returns the next higher statement.

Examples The following examples illustrate different uses for the **PGM()** function.

Example 1:

```
1030 SWAPTST:
2:
2:A$=PGM(-1)
2:B$=PGM(-2)
2:PRINT A$, " | ", B$
C:\Program Files\Sage Software\ProvideX\PGM\PVX_MAINPR | C:\Program
Files\Sage Software\ProvideX\PGM\PVX_SUBPR
```

Example 2:

```
0010 ! testpgm.-3
0020 CALL PGN+";label1;additional info"
0030 STOP
0040 LABEL1:
0050 PRINT "pgm(-3)=' ", PGM(-3), "' "
0060 EXIT
pgm(-3)='C:\Program Files\Sage
Software\ProvideX\testpgm.-3;label1;additional info'
```

Example 3:

```
0380 REM
0390 INPUT "Enter statement to display OR <F4> to Stop: ",A
0400 IF CTL<>4 THEN LET X$=PGM(A,ERR=0440) ELSE GOTO 0430
0410 PRINT LST(X$)
0420 GOTO 0390
0430 PRINT "DONE"; STOP
0440 PRINT "Error Transfer Works"; STOP
64999 PRINT "Cannot find statement"; END
-:run
Enter statement to display OR <F4> to Stop: 400
0400 IF CTL<>4 THEN LET X$=PGM(A,ERR=0440) ELSE GOTO 0430
Enter statement to display OR <F4> to Stop: 7000
64999 PRINT "Cannot find statement"; END
Enter statement to display OR <F4> to Stop: <F4> (not printable)
DONE
-:run
Enter statement to display OR <F4> to Stop: 65999
Error Transfer Works
```

POS() Function

Scan String

Format **POS(pattern\$ {= ...}string\$[,step[,instance]][,ERR=stmtref])**

Where:

- {= ...}** Relationship operator. Define the relationship for the string comparison:
 - ⋮ colon searches for any of the pattern characters in *string\$*
 - ^ caret searches for first instance of character in *string\$*, not in *pattern\$*
 - = equals sign searches for exact match
 - < > greater/less than symbols, etc.
- instance** Numeric expression tells ProvideX which occurrence(s) to report when the pattern is found in the string.
- pattern** String value or expression to scan for.
- step** Increment value of the intervals. Optional. Numeric expression.
- stmtref** Program line number or label to transfer control to.
- string\$** Character string or variable containing value to be converted to binary.

Returns Integer, starting position where relationship is satisfied (0 if none)

Description The **POS()** function scans the *string\$* to determine where a portion of it will satisfy the relationship with the pattern string. The function returns an integer reporting the starting position in *string\$* where the relationship is satisfied, or 0 *zero* if no position satisfies the relationship.

Use the *step* value to set the logical increment for the next position to be checked. The default is to move one (1) position at a time. If the value of the increment is negative, then the string is scanned from back to front.

Indicate the *instance* or occurrence for which you want to obtain the **POS()** value. If this value is omitted, the default is 1 (the first instance found). If the value is 0, the **POS()** function returns the total number of matches found.

Examples The following examples illustrate the different uses for the **POS()** function.

Example 1:

```
Given A$="The quick brown fox":
POS("q"=A$) ! yields 5
POS("z"=A$) ! yields 0
```

Example 2:

```
Given A$="The quick brown fox":
POS("q"=A$) ! yields 5
POS("z"=A$) ! yields 0
POS("o"=A$) ! yields 13
POS("o"=A$,-1) ! yields 18 - Scan from end (fox)
POS("o"=A$,1,2) ! yields 18 - Second occurrence (fox)
POS("o"=A$,2) ! yields 13 - Checks every 2nd position
POS("r"<A$) ! yields 6 - "u" is first char. > "r"
```

PRC() Function

Round Number to Precision

Format `PRC(num[,precision][,ERR=stmtref])`

Where:

num Value to be rounded. Numeric expression.

precision Precision to which to round. Optional. Numeric expression. Integer from 0 to 18. If omitted, rounding is done to the current precision in effect.

stmtref Program line number or label to transfer control to.

Returns Numeric value, rounded to a set precision.

Description The **PRC()** function returns a given numeric value (*num*) rounded to the set precision. If a *precision* parameter is supplied, the **PRC()** function rounds the value based on the new precision. If *precision* is not supplied, then the value is rounded to the current **PRECISION** in effect. .



Note: There is one exception to the above. A **ROUND ON** directive will truncate longer values to the current **PRECISION** in effect.

If you use an invalid value (e.g., >18), ProvideX returns Error #41: Invalid integer encountered (range error or non-integer).

See Also [PRECISION Directive, p.248](#),
[ROUND Directive, p.293](#)
[FLOATING POINT Directive, p.133](#)

Example

```
0010 LET A=PRC(1.3456); PRINT A,  
0020 LET A=PRC(1.3456,0); PRINT @(15),A,  
0030 LET A=PRC(1.3456,3); PRINT @(25),A,  
0040 LET A=PRC(1.3456,2); PRINT @(40),A,  
-:run
```

The results will vary, depending on current precision and rounding:

1.35	1	1.346	1.35-:	! PRECISION 2, ROUND OFF
1.35	1	1.35	1.35-:	! PRECISION 2, ROUND ON
.13456E+01	.1E+01	.1346E+01	.135E+01-:	! PRECISION -1

PRM() Function

Return Parameter Value

Format **PRM(param[,ERR=stmtref])**

Where:

param Two-character valid system parameter code, enclosed in single quotes.
See the section on accepted system parameters. String expression.

stmtref Program line number or label to transfer control to.

Returns Current value of system parameter, or status code if switch.

Description The **PRM()** function returns the current value of the specified system parameter unless the parameter is a switch. The following numeric status codes are returned for a switch:

- 0 if the switch is off,
- 1 if the switch is on,
- 1 if the specified parameter does not exist.

See Also [SET_PARAM Directive, p.306](#)

Examples This temporarily changes the **'BY'** parameter to obtain a new date:

```

0100 PRINT "Valentine days.."
0110 LET SV_BY=PRM('BY')
0120 FOR Y=1999 TO 2009
0130 SET_PARAM 'BY'=Y
0140 PRINT DTE(31+14-1:"%Dl %Ml %D/%Y")
0150 NEXT Y
0160 SET_PARAM 'BY'=SV_BY
-:run
Valentine days..
Sunday February 14/1999
Monday February 14/2000
Wednesday February 14/2001
Thursday February 14/2002
Friday February 14/2003
Saturday February 14/2004
Monday February 14/2005
Tuesday February 14/2006
Wednesday February 14/2007
Thursday February 14/2008
Saturday February 14/2009

```


PRM() returns a specific parameter's current setting (or the Boolean value for a switch):

```
->?prm('ah')  
0  
->set_param 'ah'  
->?prm('ah')  
1
```

The parameter's status is returned even when it's hidden from the **PRM** variable's contents listing:

```
->?prm('!i') ! hidden unless ON  
0
```

PTH() Function

Return Pathname

Format PTH(*chan* [,ERR=*stmtref*])

Where:

chan Channel or logical file number of file whose pathname is to be returned.

stmtref Program line number or label to transfer control to.

Returns Pathname of open file.

Description The **PTH()** function returns the operating system path of the file specified. (The value returned is an ASCII string reporting the full pathname, including directories and the filename.) If the file is a device (e.g., a printer), the device name is returned.



Note: The file must be open for the **PTH()** function to operate.

Example

With "C:\Documents and Settings\Default User\Application Data\ar" as current directory:

```
0010 OPEN (26)"PRODFL"
0020 PRINT "Just opened file: ",PTH(26)
0030 ....
```

RUN

```
Just opened file: C:\Documents and Settings\Default
                 User\Application Data\ar\PRODFL
```

PTH() returns the device name when the file is a device:

```
-:OPEN (30)PRINTER$
```

```
-:?PTH(30)
```

```
LPT1
```

PUB() Function

List Public Programs

Format **PUB(index[,ERR=stmtref])**

Where:

index Index in the **ADDR** table for the entry to be returned. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns String, addressed program information.

Description The **PUB()** function returns a string reporting the names, starting addresses and sizes of all programs for which the **ADDR** directive is active. The information is returned as a character string containing the name and description of each program addressed.

The index indicates the relative program number in the **ADDR** table. ProvideX updates the table to account for your subsequent **ADDR** and **DROP** directives and changes index numbers accordingly (They're not static.) ProvideX returns Error #41: Invalid integer encountered (range error or non-integer) if no entry in the **ADDR** table exists for your given index.

The table below lists the contents of the string returned by the **PUB()** function:

<i>Byte</i>	<i>Contents</i>
1 to 2	Program size in bytes
3	\$01\$
4 to 16	Reserved for ProvideX use
17 +	Full Pathname to program

See Also [ADDR Directive, p.30](#),
[DROP Directive, p.95](#).

Examples

```

0010 LET A$=PUB(1)
0020 PRINT A$(17) ! C:=$433A$, starts at position 17 (see below)
0030 PRINT HTA(A$)
-: run
C:\MANUALS\PVX\TEST
03BD000000080C510000000000000433A5C4D414E55414C535C5056585C54455354

```

RCD() Function

Return Next Record

Format RCD(*chan*[,*fileopt*])

Where:

chan Channel or logical file number of file to reference.

fileopt Supported file options (see also, [File Options, p.810](#)):

DOM=stmtref Missing record transfer

END=stmtref End-Of-File transfer

ERR=stmtref Error transfer

IND=num Record index

KEY=string\$ Record key value

KNO=num | name\$ File access key number (*num*) or name (*name\$*)

RNO=num Record number

Returns Contents of next or given record.

Description The RCD() function returns the contents of either the next record in the given file or of the record identified in a **KEY=**, **RNO=**, or **IND=** option. ProvideX effectively issues a **READ RECORD** directive when it encounters this function and returns the contents of the given record.

See Also [READ RECORD Directive, p.275](#)

Example

```
0010 OPEN (13)ARG(-1) ! pvx.ini
0020 PRINT RCD(13,END=40)
0030 GOTO 0020
0040 STOP
-:run
[WindowFrame]
TypeSizeLoc=1,648,485,84,340
```

RDX() Function

Convert ASCII to Radix-40

Format **RDX(ascii\$[,ERR=stmtref])**

Where:

ascii\$ Character string containing ASCII data. String expression.

stmtref Program line number or label to transfer control to.

Returns RADIX-40 equivalent of given ASCII string.

Description The **RDX()** function is used to convert data from normal ASCII to Radix-40. Conversion to Radix-40 compresses three characters (any of 0-9, A-Z, ., -, \$, or space) into two bytes of data (16 bits).

To convert data back to ASCII, use the inverse function, **TRX()**.

See Also **TRX() Convert Radix-40 to ASCII, p.542**

Example

```
0010 LET A$=LST(PGM(65000)); PRINT "string: ",QUO,A$,QUO
0020 PRINT HTA(A$)
0030 PRINT HTA(RDX(A$))
-:run
string: "65000 END "
363530303020454E4420
2CB106686E600000
```

REC() Function

Expand IOList Specification

- Formats
1. *Expand IOList from Object Code*: **REC**(*iol_obj\$* [, **REC**=*name\$*] [, **ERR**=*stmtref*])
 2. *Expand from IOList*: **REC**(**IOL**=*iolref* [, **REC**=*name\$*] [, **SEP**=*char\$*] [, **ERR**=*stmtref*])
 3. *Return Default Value*: **REC**(**FILE** *chan* [, **ERR**=*stmtref*])

Where:

- char\$* Hex or ASCII string value. Character to use as separator to parse the data; e.g., **SEP**= " . ". Dynamic **SEP**=* separators are not supported. If the **REC**() function statement also contains a **REC**= clause, the **REC**= clause must precede the **SEP**= clause.
- chan* File (channel) from which the **REC**=value will be returned.
- iol_obj\$* String expression that contains the object code of an IOList.
- iolref* Either a string variable containing the object code of an IOList or a statement reference to an IOList (statement number/label).
- name\$* Optional record name/prefix for all of the variables in the IOList. (**REC**=**VIS**(*string\$*) can also be used)
- stmtref* Program line number or label to transfer control to.

Returns String, name/contents of IOList.

Description The **REC**() function returns the name or contents of an IOList and can be used to define a record prefix.

Format 1: Expand IOList from Object Code

REC(*iol_obj\$* [, **REC**=*name\$*] [, **ERR**=*stmtref*])

With this format, the string returned by the **REC**() function is expanded from a string expression consisting of the object code of an IOList.

Format 2: Expand from IOList

REC(**IOL**=*iolref* [, **REC**=*name\$*] [**SEP**=*char\$*] [, **ERR**=*stmtref*])

The string returned by the **REC**() function in this format is expanded from a variable list (**IOL**=*iolref*).

In both the above formats, you can add a **REC**= clause to assign a record name as a prefix to all the variables in the list (similar to a prefix for a composite string variable). That is, the variable you identify in your string variable defines a prefix for the variable names in your IOList. Both formats also allow you to have other

functions and directives return values based on a variable name for the record, e.g.,
LEN (X\$) and PRINT X\$.

You can also use a **SEP=char\$**. Note that if you include both this and a **REC=** clause, the **REC=** clause must precede the **SEP=** clause.

Example:

In the following example, ProvideX would place the input data in CUST.NAME\$, CUST.ADR1\$ and CUST.ADR2\$ using the IOList at line 0110 as a template (even though there is no matching prefix in the IOList). That is, you can use the same IOList as a template for different **REC=** prefix specifications.

```
0100 PRINT X$
0110 IOLIST NAME$,ADR1$,ADR2$
0120 INPUT EDIT "Name",@(9)," ": ",CUST.NAME$
0130 INPUT EDIT "Address 1: ",CUST.ADR1$
0140 INPUT EDIT @"(8),"2: ",CUST.ADR2$
0150 LET X$=REC(IOL=0110,REC=CUST$)
0160 IF LEN(X$)>100 THEN PRINT "TOO-LONG"; GOTO 0120
0170 PRINT X$
-:run
BRETT'S BODYSHOP
123 SOME ST.
ANYTOWN SK SOM 0V0
```

The values are also displayed with the prompts below. The user edits to change them:

```
Name      : YVONNE'S BODYSHOP
Address 1: 123 SOME RD.
           2: NEWTOWN SK SOM 0V0
YVONNE'S BODYSHOP
123 SOME RD.
NEWTOWN SK SOM 0V0
```

Format 3: Return Default Value

REC(FILE chan[,ERR=stmtref])

Use this format to obtain the name of the **REC=** default value for your given file.

REF() Function

Control Reference Count

Formats REF({ADD|DROP|READ} *obj_id*)

Where:

ADD Keyword indicating an increment of the reference count for *obj_id*.

DROP Keyword indicating a decrement of the reference count for *obj_id*.

obj_id Object Identifier

READ Keyword indicating no change to the reference count for *obj_id*.

Returns Current reference count value (0 if the object is deleted).

Description The **REF ()** function is used in *Object Oriented Programming* to control access to an object by incrementing or decrementing the reference count. The **DROP OBJECT** directive can also be used to destroy an object.

The system sets the reference count for any newly created object to 1 (one). If you plan to use an object more than once, use **REF (ADD *obj_id*)** to increment the reference count.

When finished with an object, use **REF (DROP *obj_id*)** to decrement the reference count. When the reference count becomes zero, the object is deleted.

Only objects whose reference count is 1 can be deleted. Once an object is destroyed, its identifier may be re-assigned to another object. You should no longer use the object identifier in your application, as it may reference a totally different object (if the identifier is re-assigned).

All objects are destroyed when the application issues a **START** directive or if the **END** directive is entered at command mode.

REF (READ *obj_id*) returns the current reference count value. All calls to **REF ()** return the current reference count (or 0, if deleted).

See Also [DEF CLASS Directive, p.65](#)
[LOAD CLASS Directive, p.195](#)
[DROP CLASS Directive, p.102](#)
[RENAME CLASS Directive, p.283](#)
[STATIC Directive, p.329](#)
[DROP OBJECT Directive, p.104](#)
[NEW\(\) Create new Object, p.489](#)
[Data Integration, User's Guide](#)

RND() Function

Return Random Number

Format **RND(seed[,ERR=stmtref])**

Where:

seed Numeric expression must be (or result in) an integer. The value of this number is used to determine the result of the function.

stmtref Program line number or label to transfer control to.

Returns Random numbers based on given value.

Description The **RND()** function returns random numbers based on the *seed* given. The following table describes the value returned based on the seed value:

- > 0 If *seed* is greater than 0 zero, the random number returned will be in the range of zero to one less than the given number (to a maximum of 32768).
- = 0 If *seed* equals 0 zero, the random number returned will be between 0 and 1, with PRECISION=8.
- < 0 If *seed* is less than 0 zero, **RND()** initializes the seed for generating the next sequence of random numbers, similar to use of the **RANDOMIZE** directive.

When *seed* is 0 or greater, each call to the **RND()** function will yield a different random number. Executing the **RANDOMIZE** directive and using the **RND()** function with a negative value will produce similar results; e.g.,

```
?RND(-10) = RANDOMIZE 10 = RANDOMIZE 10
             ?RND(0)      = ?RND
```

For further randomization, use **RND(-TME)** as a method to re-initialize the seed value.



Note: ProvideX returns an error when *seed* is greater than 2147483647 or is not a valid integer.

See Also [RANDOMIZE Directive, p.270](#)
[RND System Variable, p.571](#)

```
Example
0010 REM
0020 FOR I=1 TO 10
0030 PRINT RND(9),
0040 NEXT
0050 PRINT 'LF',RND(0)
0060 PRINT RND(-1)
0070 PRINT " DONE"; END
-:run
1 1 7 0 7 7 4 4 5 7
0.05947216
0.11337858
DONE
```

RNO() Function

Return Next Record Number

Format **RNO(chan[,fileopt])**

Where:

chan Channel or logical file number of the file to reference.

fileopt Supported file options (see also, [File Options, p.810](#)):

END=stmtref END-OF-FILE transfer

ERR=stmtref Error transfer

IND=num Record index

KEY=string\$ Record key

KNO=num | name\$ File access key number (*num*) or name (*name\$*)

Returns Integer, position of next/given record.

Description The **RNO()** function reports the position in the file specified of either the next record, or the record identified in the **KEY=** or **IND=** options. The **RNO()** value is the absolute ordinal position of the record in the file. The first record in a file returns a value of 1 (one), the second **RNO()** value is 2, and so on. In a Keyed file, the record number depends on the key chosen and its value relative to all other records in the same file.



Note: ProvideX supports the **RNO()** function for ODBC files. See *File Handling* in the *ProvideX User's Guide*.

Example

```

0010 OPEN (13)"PVX_KEYD"
0020 INPUT "Which record? ",@(15),K$,
0030 IF K$="" THEN CLOSE (13); PRINT "DONE"; STOP
0040 READ (13,KEY=K$,ERR=0100)
0050 PRINT " Key ",K$," is Rec# ",RNO(13,END=0130)
0060 GOTO 0020
0100 REM 100
0110 PRINT " is invalid",@(40),"...Please try again"
0120 GOTO 0020
0130 PRINT @(22),"...Sorry...END-OF-FILE"
0140 END
-:end
-:run
Which record? ABCDEF is invalid ...Please try again
Which record? 123456 Key 123456 is Rec# 2
Which record? 123460 Key 123460 is Rec# 6
Which record? 123458 Key 123458 is Rec# 4
Which record? DONE ! User hit <Enter>
-:run
Which record? 123461 ...Sorry...END-OF-FILE

```

SEP() Function

Return Field Separator

Format **SEP**(*chan*)

Where:

chan Channel or logical file number of an **OPEN** file.

Returns Either the **SEP** value (single character string) for an **OPEN** file on given channel or "" (null).

Description The **SEP()** function returns either a single character string, **SEP** (the field separator value in hex) for a given file **OPEN** on a given channel, or "" (null).
The value returned is the value you set in the **SEP=** option when creating a Keyed or Indexed file (a single character string which can be anything from \$00\$ to \$FF\$).
The function returns "" (null) for a dynamic field separated file (a Keyed or Indexed file created using a **SEP=*** option). With this type of file, ProvideX stores each field in the record with a one-byte prefix identifying the type and length of data that immediately follows.

Example ->open (14) "pvx_dir"
 ->?hta(sep(14))
 8A

SGN() Function

Return Sign of Value

Format **SGN(num[,ERR=stmtref])**

Where:

num Value whose sign is to be returned. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Status code, 0 zero, 1 or -1.

Description The **SGN()** function returns a numeric status code for the sign of a value:

1 If *num* is greater than zero.

0 If *num* equals 0 zero.

-1 If *num* is less than zero.

Example

?SGN(23.492) ! yields 1

?SGN(4-4) ! yields 0

?SGN(4-7.34) ! yields -1

SIN() Function

Sine Function

Format **SIN(num[,ERR=stmtref])**

Where:

num Value whose sine is to be returned. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Numeric, range -1 to 1.

Description The **SIN()** function returns the sine of the numeric expression specified. It will return a numeric value between -1 and 1, rounded to the current **PRECISION** in effect. Its inverse function is **ASN()**.

Example

```
00010 PRINT 'CS'
00020 X=0,Y=80
00030 LASTX=0, LASTY=80
00040 FOR I=-15 TO 1 STEP .5
00050 X=X+25
00060 Y=50*SIN(I)+80
00070 PRINT 'LINE' (LASTX, LASTY, X, Y)
00080 LASTX=X
00090 LASTY=Y
00100 NEXT
```

SQR() Function

Square Root

Format **SQR(num[,ERR=stmtref])**

Where:

num Value whose square root is to be returned. It must be a positive number. Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Numeric, square root of given value.

Description The **SQR()** function returns the square root of the number provided, rounded to current **PRECISION**. If you use an invalid value (e.g., a negative value such as `SQR(-2.345)`), ProvideX returns an Error #40: Divide check or numeric overflow.

Example

```

0010 INPUT "Length of one side: ",A
0020 IF A<=0 THEN GOTO 0010
0030 INPUT "Length of other : ",B
0040 IF B<=0 THEN GOTO 0030
0050 PRINT "Hypotenuse is : ",SQR(A*A+B*B,ERR=*NEXT)
0060 PRINT "DONE"; STOP
-:run
Length of one side: 2.3
Length of other   : 1.65
Hypotenuse is    : 2.83

```

SRT() Function

Sort String

Formats

1. *Sort Equal Length Elements*: **SRT(string\$,fixed_len[,ERR=stmtref])**
2. *Sort Delimited Elements*: **SRT(string\$[,dlim_char\$][,ERR=stmtref])**

Where:

- dlim_char\$** Delimiter for each string element. If you omit this, the default delimiter is the **SEP** character (e.g., \$8A\$).
- fixed_len** Numeric expression. Length of each element in the string (all the same fixed length).
- stmtref** Program line number or label to transfer control to.
- string\$** String to be sorted.

Returns

Sorted character string.

Description

The **SRT()** function returns a sorted character string. The string elements are internally sorted into ascending sequence. The order of duplicate elements is undefined.



Note: To sort a string into descending order, use the **NOT()** function to invert the bits in the string prior to and after sorting. (See **NOT() Invert String Bits/Logical Condition**, p.490.)

Format 1: Sort Equal Length Elements

SRT(string\$,fixed_len[,ERR=stmtref])

If each element in the string is the same fixed length, use this format to sort the string.

Examples:

```
0100 REM To sort in ascending order:
0110 A$="0231405242670291234403242"
0120 PRINT SRT(A$,5)
->RUN
0231403242052421234467029
0200 REM To sort in descending order:
0220 PRINT NOT(SRT(NOT(A$),5))
->RUN
6702912344052420324202314
```

Format 2: Sort Delimited Elements

SRT(string\$[,d1m_char\$][,ERR=stmtref])

If each element in the string has a delimiter or **SEP** character, use this format to sort the string. ProvideX considers the first character of *d1m_char\$* to be your delimiter. (The default is the current **SEP** value.)

Example:

```
0300 REM To sort with delimited strings:
0310 A$="THE BROWN FOX JUMPED OVER THE DOG " ! Blank as delimiter, ends string
0320 PRINT SRT(A$, " ")
->RUN
BROWN DOG FOX JUMPED OVER THE THE
```


SSZ() Function

Return Sector Size

Format **SSZ**(*drive* [, **ERR**=*stmtref*])

Where:

drive Disk drive whose sector size is to be returned (ignored). Numeric expression.

stmtref Program line number or label to transfer control to.

Returns Always 256.



Note: This function is included for compatibility with other languages.

Description The **SSZ()** function has been provided for compatibility with Business Basic systems. In ProvideX, it always returns 256.

STK() Function

Program Call Stack

Format **STK(level[,ERR=stmtref])**

Where:

level Program **CALL** stack level. Numeric expression.

string\$ String expression to be processed.

Returns String, program line number (characters 1 to 5) plus full pathname.

Description The **STK()** function returns a character string reporting the program line number (5 digits) and program (full pathname) being executed at the program level specified.

Use a negative value to specify a level that is relative to the current level. ProvideX returns an Error #41: Invalid integer encountered (range error or non-integer) if your given level does not exist.

Example The current directory is "/usr/test". In program "PROG1":

```
0100 CALL "STACKER"
```

In program "/usr/test/STACKER"

```
0010 PRINT "Level 1=",STK(1)
```

```
0020 PRINT "Level 2=",STK(2)
```

```
0030 PRINT "Level -1=",STK(-1)
```

When run:

```
Level 1=00100/usr/test/PROG1
```

```
Level 2=00020/usr/test/STACKER
```

```
Level -1=00100/usr/test/PROG1
```

The next example prints all the programs in the stack:

```
0010 I=1-PRM('B0')
```

```
PRINT STK(I,ERR=*NEXT); I=I+1; GOTO 20
```



Note: You can use PRM('B0') to circumvent a potential problem should the base level be level 0, rather than 1.

STP() Function Strip Leading/Trailing Characters

- Formats
1. *Strip Character from a String*: **STP(string\$, [stp_code[\$]][, [*]stp_char\$][, ERR=stmtref])**
 2. *Remove Mnemonic from a String*: **STP(MNEMONIC string\$[, ERR=stmtref])**

Where:

- * Asterisk indicates a list of characters to strip. For example, STP(X\$, 1, "*" , : ") strips any trailing spaces, commas, or colons from X\$. A null string results in an Error #46: Length of string invalid.
- stp_char\$ Characters to be stripped from string. If omitted, blanks are stripped; e.g., STP(PTR\$, 3, \$1B\$) ! strips out all escape characters
STP("Hello there", 3, "e") ! strips out every lower-case "e"
- stp_code[\$] Strip code, either numeric or string:
 - 0 or L Strip Left (leading characters)
 - 1 or R Strip Right (trailing characters) - *default*.
 - 2 or B Strip Both Left and Right, and leave center (C)
 - 3 or A Strip All occurrences.
- stmtref Program line number or label to transfer control to.
- string\$ String expression to be processed.

Returns Stripped character string.

Description The **STP()** function returns a character string generated by stripping specified instances of a character, or a mnemonic, from a string expression.

Format 1: Strip Character from a String

STP(string\$, stp_code[\$][, [*]stp_char\$][, ERR=stmtref])

This format strips the character *stp_char\$* (spaces if omitted) from a string expression. Depending on the *stp_code* value, the data can be stripped from the beginning of the *string\$*, the end of the *string\$*, or from both the beginning and end. **STP()** can also be used to strip all occurrences of *stp_char\$* within the string.

Given A\$ = " This is a test "	STP() Returns
STP(A\$, 0)	"This is a test "
STP(A\$, 1)	" This is a test"
STP(A\$, 2)	"This is a test"
STP("000Test00", 2, "0")	"Test"
STP(A\$, 3)	"Thisisatest"

Format 2: Remove Mnemonic from a String

STP(MNEMONIC string\$[, ERR=stmtref])

This format removes all mnemonics contained within the *string\$*.

Example

The following code sample illustrates stripping of *left*, *right*, *both*, *all* as well as the use of "*" to indicate a list of characters:

```

0100 ! ^100 - STP function
0110 Orig$="...Test...String...",Char$="."
0120 print 'LF',"Original String: "+@(21)+'BR'+Orig$+'ER'+'LF'
0130 Strip=0,Strip$="L"; gosub StripIt; print
0140 Strip=1,Strip$="R"; gosub StripIt; print
0150 Strip=2,Strip$="B"; gosub StripIt; print
0160 Strip=3,Strip$="A"; gosub StripIt
0170 !
0200 ! ^100 - Strip multiple characters
0210 Orig$="xxx Test.zzz String yyy"
0220 print 'LF',"Original String: "+@(25)+'BR'+Orig$+'ER'
0230 !
0240 Char$="xyz.",Strip=3,Strip$="A"
0250 print "STP(Orig$,"+quo+Strip$+quo+",*"+quo+Char$+quo+") = ",
0260 print 'BR'+stp(Orig$,Strip$,*Char$)+'ER'
0270 stop
0280 !
0300 ! ^100
0310 StripIt:
0320 print "STP(Orig$,"+pad(str(Strip),3,2)+",""+quo+Char$+quo+") = ",
0330 print @(21)+'BR'+stp(Orig$,Strip,Char$)+'ER'
0340 print "STP(Orig$,"+quo+Strip$+quo+", "+quo+Char$+quo+") = ",
0350 print @(21)+'BR'+stp(Orig$,Strip$,Char$)+'ER'
0360 return

```

STR() Function

Convert Numeric to String

- Formats**
1. *Convert Numeric String to ASCII*: **STR**(num[:mask\$][,err_val\$][,ERR=stmtref])
 2. *Convert ASCII String to Mask*: **STR**(string\$:mask\$[,err_val\$][,ERR=stmtref])
- Where:**
- err_val\$** Error value to return should the conversion fail. String expression. If you omit **err_val\$** and the conversion fails, ProvideX either reports **Error #43: Format mask invalid** or returns the value unformatted, depending on your setting for **'FI'**.
- mask\$** Format mask to be used in the conversion process. Max string size 8kb. For a list of valid mask characters, refer to **Data Format Masks , p.813**.
- num** Numeric value to convert to a character string.
- string\$** String expression to be processed.
- stmtref** Program line number or label to transfer control to.

Returns String, converted and validated.

Description The **STR()** function converts and validates strings. Add a format mask to specify the size and format of the resultant character string. When a format mask is included, **STR()** returns a string value converted from numeric using the default decimal point and thousands separator set by **'DP'** and **'TH'**. If the format mask is omitted, **'DP'** and **'TH'** settings are ignored.

See Also **'FI'** System Parameter, *p.666*,
'DP'= System Parameter, *p.662*,
'TH'= System Parameter, *p.690*,
Data Format Masks , p.813.

Format 1: Convert Numeric String to ASCII

STR(num[:mask\$][,err_val\$][,ERR=stmtref])

Use this format to convert a numeric value to an ASCII character string; e.g.,

A\$=STR(5*6) ! (yields A\$="30")

A\$=STR(5*6:"000") ! (yields A\$="030")

A\$=STR(.01:"0.00") ! (yields A\$="0.01")

A\$=STR(99:"0.00","*****") ! (yields A\$="*****")

Format 2: Convert ASCII String to Mask

STR(string\$:mask\$[,err_val\$][,ERR=stmtref])

Use this format to convert and validate a *string* value based on the format *mask* (which dictates the size and content of the results) e.g.,

```
A$=STR("1234567":"000-0000") ! (yields A$="123-4567")
```

```
A$=STR("AB":"00","**") ! (yields A$="**")
```

Examples

The following will print ****OverFlow**** if the value is too large for the mask:

```
x=1234567890.12
```

```
PRINT STR(x:"####,##0.00-", "**OverFlow**") ! Yields **OverFlow**
```

If value is too large with pennies, we'll try it without the pennies, or else print ****OverFlow****:

```
x=1234567890.12
```

```
PRINT STR(x:"####,##0.00-",str(x:"####,###,##0", "**OverFlow**")) ! Yields  
1234,567,890
```

SUB() Function*Substitute Text*

Format **SUB**(*string*,\$,*search*,\$,*replace*,\$[,*instance*])[,**ERR**=*stmtref*])

Where:

instance Optional. Integer. Which occurrence to find:

- 0 All occurrences (assumed if not given).
- >0 Specific occurrence, searching left to right within *string*%; i.e.,
1=first occurrence, 2=second occurrence ...
- <0 Specific occurrence, searching from right to left; i.e., -1=first
occurrence from end of *string*%, -2=second occurrence from
end of *string*% ...

search,\$ Value for which to search. String expression. Cannot be null.

replace,\$ Text to replace the original *search* text, if found. String expression. You
must include a *replace* value, but the value can be null; e.g.,
SUB(A\$,B\$, "", 0, ERR=100).

stmtref Program line number or label to transfer control to.

string,\$ String expression containing the string in which to search-and-replace
and perform the substitution.

Returns Converted string.

Description The **SUB()** function performs a substitution within a string. The *search* value can't
be null. (That generates an Error #46: Length of string invalid.) If no
substitutions occur, no error is generated.

Example

```
0100 PRINT 'CS'; LIST
0110 LET A$="Hello there",B$="e",Z$=SUB(A$,B$,"",0)
0120 PRINT Z$+" "+A$
0130 END
-: run
Hllo thr Hello there
```

SWP() Function

Swap Data

Format `SWP(string$[,swp_code][,ERR=stmtref])`

Where:

stmtref Program line number or label to transfer control to.

string\$ String expression whose bytes are to be swapped.

swp_code Type of byte-swapping to be performed. Numeric expression. See the chart below.

Returns Conversion of data, native machine format to/from ProvideX common format.

Description The **SWP()** function is designed to simplify the conversion of data between native machine format and ProvideX common format. ProvideX processes data with the most significant information to the left (i.e., you read the data from left to right) where some computers store data with the most significant data at the right.

The **SWP()** function gives you a convenient way of rearranging the data so it can be exchanged. If you do not supply a swap type, the function will rearrange the data according to the native operating mode of the machine. For example, on INTEL 80x86 CPUs, a **SWP()** of "12345678" yields "87654321".

Unfortunately not all swapping algorithms are this easy. Some computers swap every two characters, some every four characters and some every eight characters. The following table describes the various types of swapping based on a value of "12345678".

<i>Type</i>	<i>Value</i>	<i>Type</i>	<i>Value</i>
0	12345678	4	56781234
1	21436587	5	65872143
2	34127856	6	78563412
3	43218765	7	87654321



Note: For compatibility mode, the swap type can be provided as a single byte containing the binary value of the desired type; i.e., \$01\$, \$02\$, ...

SYS() Function *Invoke Operating System Command*

- Formats**
1. *Invoke Operating System Command*: **SYS(command\$[,ERR=stmtref])**
 2. *Signal Another Process (PVX UNIX/Linux Only)*: **SYS(pid[,signal][,ERR=stmtref])**

Where:

- command\$** Command to be processed. Maximum string size 8kb.
- pid** UNIX Process ID value to check/notify. Numeric expression.
- signal** Optional signal value to send to the UNIX Process ID. Numeric expression.
- stmtref** Program line number or label to transfer control to.

Returns Operating system code identifying command passed to it by the function.

Description The **SYS()** function passes a given string or numeric Process ID to the operating system command processor for execution.

Format 1: *Invoke Operating System Command*

SYS(command\$[,ERR=stmtref])

SYS() returns the operating system's code identifying the command. It returns zero if the task is running and not zero (usually -1) if it is unsuccessful. The value in the system variable **RET** can be checked for an error value if the function was unsuccessful.

Example:

```
0010 PRINT 'CS', "Menu"
0020 PRINT " 1: List directory"
0030 PRINT " 2: Run WORD processor"
0040 PRINT " 3: Run SPREADSHEET"
0050 PRINT " X: Sign off"
0060 INPUT "Enter selection:", X$
0070 IF X$="1" THEN LET X$="ls"; GOTO 0500
0080 IF X$="2" THEN LET X$="wp"; GOTO 0500
0090 IF X$="3" THEN LET X$="123"; GOTO 0500
0100 IF X$="X" THEN QUIT
0110 PRINT 'RB', ; GOTO 0060
0500 A=SYS(X$)
0510 IF A<>0 THEN PRINT "Command '"+X$+"' FAILED"
0520 GOTO 0020
```

Format 2: Signal Another Process (PVX UNIX/Linux Only)**SYS(pid[,signal][,ERR=stmtref])**

You can send a signal from another ProvideX session or from the UNIX shell to generate a CTL event using a `SIGNAL 2` (a `SIGBREAK` that initiates a program's **SETESC** branch) and you can also send signals from inside a ProvideX program. The `GID` variable contains the binary value of the UNIX Process ID, `pid=dec(00+gid)`. Use this value with the **SYS()** function to create a signal to another process:

```
X=SYS(pid,2)
```

where the first argument is the PID signal to send and the second is the signal number. `TCB(89)` returns the numeric value of the current Process ID. Use this as a replacement for `DEC(00+GID)` to retrieve the current PID value.

See Also

INVOKE Execute Operating System Command, p.163
GID Operating System Process Identifier, p.562

TAN() Function

Return Tangent

Format **TAN(num[,ERR=stmtref])**

Where:

num Numeric expression whose Tangent will be returned.

stmtref Program line number or label to transfer control to.

Returns Numeric, rounded, Tangent of given number.

Description The **TAN()** function returns the Tangent of a given numeric expression. The numeric value returned will be rounded to the current **PRECISION** in effect. The inverse function is **ATN() Return Arc-Tangent, p.399**.

Examples FLOATING POINT
-:PRINT TAN(1)
 .15574077246549E+01
-:PRINT TAN(-3.14159/2)
 -.75369599539306E+06

TBL() Function

Convert String Via Table

- Formats
1. *Translation Table in Program*: **TBL**(string\$,TBL=tbl_stmtref[,ERR=stmtref])
 2. *Translation Table in Variable*: **TBL**(string\$,tbl_var\$[,ERR=stmtref])
 3. *Translate Using Position*: **TBL**(position,expr_0[\$],expr_1[\$] ... ,expr_n[\$][,ERR=stmtref])
 4. *Character to Character Conversion*: **TBL**(var\$,compare\$,table\$)

Where:

<i>compare\$</i>	String table to compare character by character with the value in <i>var\$</i> . String expression.
<i>expr_0[\$]...n</i>	List of expressions to be returned. Numeric or string expressions. Restriction: The expressions must all be the same type (i.e., all characters or all numerics).
<i>position</i>	Determines which expression to use. Positional or numeric expression, range 0 zero to <i>n</i> .
<i>stmtref</i>	Program line number or label to transfer control to.
<i>string\$</i>	String to be translated/replaced (must be all characters or all numeric).
<i>table\$</i>	Table to use for conversion when a character in the compared value matches the value in <i>var\$</i> . String expression.
<i>tbl_stmtref</i>	Optional, if your table is embedded in the same statement as the TBL() function. If you refer to a program <i>tbl_stmtref</i> , it must contain a conversion table.
<i>tbl_var\$</i>	String variable. Contains conversion table to replace elements in the <i>string\$</i> . String expression.
<i>var\$</i>	String variable. Contains a string to be translated or replaced.

Returns Converts string to values in table.

Description The **TBL()** function converts a string (all characters or all numeric) to the corresponding values set in a given translation table.

See Also *Translation Tables* in the *ProvideX User's Guide*

Format 1: Translation Table in Program

TBL(string\$,TBL=tbl_stmtref[,ERR=stmtref])

Use this **TBL()** format if the translation table to convert the *string\$* is an embedded table in your program. If you include a *tbl_stmtref*, it must refer to a program statement containing a conversion table defined by a **TABLE** directive. This statement reference is optional if the conversion table is embedded in the same statement as the **TBL()**function. For more information, refer to the **TABLE Directive**, p.340.

Format 2: Translation Table in Variable

TBL(string\$,tbl_var\$[,ERR=stmtref])

Use this format if the translation table to convert the *string\$* is in a variable.

Format 3: Translate Using Position

TBL(position,expr_0[\$],expr_1[\$] ... ,expr_n[\$][,ERR=stmtref])

Use this format to obtain a value based on the numeric value of an expression. The value the **TBL** () function returns depends on the value of position. If it's 0 *zero*, then the value of *expr_0* will be returned, if 1, then *expr_1* will be returned, and so on.



Note: The values for *expr_0[\$]*, *expr_1[\$]*, ...*n[\$]* can be either string or numeric. The only requirement is that they must be *all* the same: *all* string or *all* numeric.

Format 4: Character to Character Conversion

TBL(var\$,compare\$,table\$)

Use this format to do a character-to-character conversion of a string. ProvideX replaces each character in the string variable that matches a character in the first comparison string expression with the corresponding character from the table. That is, when a character in the string variable matches a character in *compare\$*, then:

- *compare\$* character 1 is replaced by *table\$* character 1 in *var\$*,
- *compare\$* character 2 is replaced by *table\$* character 2 in *var\$*,
- *compare\$* character 3 is replaced by *table\$* character 3 in *var\$*,
- and so on.

Examples

Example 1:

```
T$="W"
P$=TBL(POS(T$="DWM"), "???", "Daily", "Weekly", "Monthly")
?P$
Weekly
```

Example 2:

```
PRINT "Expires:", TBL(EXP_DT$="", EXP_DT$, "Never")
```

If *EXP_DT\$* is null, the logical expression *EXP_DT\$=""* yields 1. A null date will print "Never". Otherwise, it yields 0 *zero* and the value of *EXP_DT\$* is printed. In effect, this form of **TBL**() becomes: **TBL**(*logical_expr*, *else_value*, *true_value*).

Example 3:

```
let string$="ABCDEFGH 123"
newstring$=tbl(string$,"ACF ", "123_") ! newstring$ is "1B2DE3G_123"
```

TCB() Function

Return Task Information

Formats

1. *Return Task Information*: **TCB**(*tcb_code*[,**ERR**=*stmtref*])
2. *Return Activation Information*: **TCB**(*pkg_index*,*info_reqd*[,**ERR**=*stmtref*])
3. *Return Platform/User Information*: **TCB**(*keyword* [,*arg*] [,**ERR**=*stmtref*])

Where:

- arg* Argument associated with *keyword* (if required).
- info_reqd* Any of the following values: 0 - actual package number, 1 - expiry date, 2- package activation flags.
- pkg_index* A number from 0 to the number of packages installed.
- keyword* Keyword requesting specific platform or user information.
- stmtref* Program line number or label to transfer control to.
- tcb_code* Numeric code indicating what type of information is to be returned.

Description

The **TCB()** function returns information for several different purposes. A single *tcb_code* specified in the **TCB()** function returns a corresponding system control value – see **Format 1: Return Task Information**. Two numeric codes (*pkg_index*,*info_reqd*) are required for activation/licensing information – see **Format 2: Return Activation Information**. Specific *keywords* are used to return specific information from the OS - See **Format 3: Return Platform/User Information**.

Format 1: Return Task Information

TCB(*tcb_code*[,**ERR**=*stmtref*])

When **TCB()** is used with a *single code* (*tcb_code*), it returns task information in the form of a numeric system control value. The codes and their values are listed below:


TCB() Return Values

- 0 Zero (0) - Reserved
- 1 Zero (0) - Reserved
- 2 Zero (0) - Reserved
- 3 Internal system error code.
- 4 Current statement number. (In a running program, not Command mode.)
- 5 The line number of the last error.
- 6 **SETESC** statement number.
- 7 **SETERR** statement number.
- 8 Top statement number in **GOSUB** stack.
- 9 Zero (0) - Reserved for future use.

TCB() Return Values (continued)

- 10 Internal system error code.
- 11 Retry statement number.
- 12 Current level number.
- 13 Current level number – ProvideX includes current level in both **TCB(12)** and **TCB(13)** for compatibility with other Business Basics.
- 14 Current precision (-1=Scientific Notation). For more information, refer to the **PRECISION Directive, p.248**.
- 15 Current maximum memory size in bytes.
- 16 Length of string found in last **MSK()** function.
- 17 Current level of user functions.
- 18 Current iteration of range assignment.
- 19 Current iteration for enhanced **FOR..NEXT** loop.
- 20 Number of arguments in **CALL**.
- 21 Current activation flags.
- 22 Number of days left for current activation.
- 23 Maximum user count.
- 24 Number of users available.
- 25 Operating system version number.
- 27 Reports current session's user slot number and type of slot:
 - > 0 Dedicated non-shared user slot number being used.
 - < 0 Shared user slot number being used.
 - 0 Task is not being counted in the user slot table (background task).
 (For further information, refer to the **'1U' System Parameter, p.655**.)
- 29 Reports the current build level of ProvideX.
- 30 Statement number of last error in called program.
- 31 Last object to lose focus.
- 32 Version level of activation.
- 33 System serial number.
- 34 System machine class.
- 35 System identification number.
- 36 ProvideX Library version code.
- 37 EFF support:
 - 0 EFF not supported
 - 1 EFF support for files up to 2GB
 - 2 EFF support for files over 2GB.
- 39 Highest used object identifier. (0 = no objects in system).
- 40 **SETTRACE** file number (if active).
- 41 Number of duplicate labels on last save.
- 42 Number of unknown state labels on last save.

TCB() Return Values (continued)

- 43 Offset to the last detected error in the last statement compiled. Reports the number of characters before the error occurred.
- 44 Number of seconds of offset to GMT (Greenwich Mean Time).
- 45 Daylight Savings indicator: 1 if Daylight Savings active, 0 if not.
- 50 Number of file reads.
- 51 Number of file writes.
- 52 Number of Keyed I/O forced buffer flushes.
- 53 Number of programs loaded from disk.
- 54 Number of programs loaded from program cache.
- 55 Program Swapouts.
- 56 Program Swapins.
- 57 Program reloads from disk.
- 60 Keyed file header busy retries (PVX for Windows).
- 61 Busy record count.
- 62 Number of unsuccessful file opens.
-  **Note:** For **TCB(63)** to **TCB(66)**, below, the values in the system parameters '**VR**' and '**VW**' define the number of read/write retries before a data read/write error will be generated. For more information, refer to the '**VR**'= [System Parameter, p.693](#), and the '**VW**'= [System Parameter, p.693](#).
- 63 Number of **READ**s verified.
- 64 Number of **WRITE**s verified.
- 65 Number of **READ** mis-compares.
- 66 Number of **WRITE** mis-compares.
- 67 On completion of **KEYED LOAD** command:
 > 0 the number of keys reloaded.
 - 1 Keyed load encountered different number of keys on different key chains.
- 68 Password retry count.
- 70 Number of logical **OPEN** directives executed.
- 71 Number of logical **READ/EXTRACT/FIND** directives executed.
- 72 Number of logical **WRITE/REMOVE** directives executed.
- 73 Number of dynamically added EFF file buffers.
- 74 Number of common buffers added dynamically for VLR/FLR files.
- 80 Last VBX status or error code. See the [CUSTOM_VBX Directive, p.61](#) for a list of codes and explanations.
- 81 Last Window LPARAM= value.

TCB() Return Values (continued)

- 82 Windows OS version:
 0 if running under Windows 3.1.
 1 if running under Windows 95.
 2 if running under Windows Server.
 -1 if running 16-bit PVX.
- 83 Status of the debug window:
 \$01\$ - Command Window is active
 \$02\$ - Trace Window is active
 \$04\$ - Break Window is active
 \$08\$ - Watch Window is active
- 84 Exit status of any child process created with an **INVOKE** or a **SYS()** or **OPEN** of a pipe. Valid for **INVOKE** or a **SYS()** if checked immediately after. Valid for **OPEN** only if wait ('**WP**' system parameter) is on the close.
- 85 Process ID of the child process created from an **INVOKE** or **SYS()** or **OPEN** of a pipe (<>|). This is valid until the next **INVOKE**, **SYS()** or **OPEN** of a pipe.
- 86 Serial number of WindX client.
- 87 **UNIX/Linux Only.** Returns PID of lock conflict. When a file I/O attempt is made and fails due to a lock, TCB(87) returns the PID of the process owning the lock. Use this process ID along with the output of a 'ps' command to identify who has a record locked.
- 88 WindX version number or 0 zero if not running under a WindX session.
- 89 Numeric value of current PID. This allows for PID values which are larger than the 16-bit values returned by GID, and will remain constant for 32-bit, upcoming 64-bit (and beyond) operating systems. Replaces the use of DEC(\$00\$+GID) to retrieve the current PID value.
- 91 Current Task/Thread Priority level. Three system parameters control the priority of a task, primarily to balance the load in a client-server environment. See '**Q_**', '**Q^**' and '**QF**' Task Priorities, p.682.
- 92 Yields the **TIM= value** * 100 (100^{ths} of a second) when doing embedded I/O, or -1 if no **TIM=**.
- 93 Current value of logical "." variable. See **WITH..END WITH Directive**, p.382.



Note: TCB(94) to TCB(97) apply to UNIX/Linux only.

- 94 User ID Number the instance of ProvideX started with.
- 95 Group ID Number the instance of ProvideX started with.
- 96 Current User ID Number.
- 97 Current Group ID Number.
- 99 Program security flags.
- 100 High memory in use (Legacy DOS only).
- 101 Number of times variables were referenced.
- 102 Number of variables not found in the variable table.

TCB() Return Values (continued)

- 103 Number of memory compares performed to locate variables.
- 104 Number of times the variable table was re-balanced.
- 110 Number of memory allocations made by ProvideX.
- 120 ID of the COM object that fired this event.
- 121 Number of COM events that have been dispatched to ProvideX.
- 122 Current depth of the event call stack.
- 123 Number of COM events that have been discarded by ProvideX due to excessive outstanding event calls.
- 170 NT Service handle. 0 if not running as a service.
- 191 LibHaru Support: 1 if available, 0 if not.
- 195 ZLib Support: 1 if available, 0 if not.
- 196 **DLL()** UNIX/Linux support: 1 if enabled, 0 if not.
- 197 ODBC: 1 if enabled, 0 if not.
- 198 DB2 Support: 1 if enabled, 0 if not.
- 199 Built-in SSL: 1 if SSL supported, 0 if not.
- 200 OCI available: 1 if OCI is enabled, 0 if not.
- 201 OCI Connects
- 202 OCI Opens
- 203 OCI Shares
- 204 OCI Selects
- 205 OCI Inserts
- 206 OCI Updates
- 207 OCI Deletes
- 208 OCI Prepare Selects
- 209 OCI Prepared Select Used
- 210 OCI Prepare Insert
- 211 OCI Prepared Insert Used
- 212 OCI Prepare Failures
- 213 OCI Dictionary Reads
- 214 OCI User Commands



Note: The following return information used for sizing variables when building a C structure, either a numeric size (in bytes not bits) or 0 if there is no such data type.

- 301 Size of a pointer (any kind of memory pointer)
- 302 Size of a *Boolean*

303 Size of a *char* or *byte*

TCB() Return Values (continued)

304 Size of a *short* or *short int*

305 Size of an *int*

306 Size of *long* or *long int*

307 Size of a *long double*

308 Size of a *long long*

309 Size of a *double*

310 Size of a *float*

311 Size of a WORD (MS Windows only)

312 Size of a DWORD (MS Windows only)

313 Size of a HANDLE (MS Windows only)

314 Size of a WCHAR (MS Windows only)

315 Size of a LPARAM (MS Windows only)

316 Size of a WPARAM (MS Windows only)

Format 2: Return Activation Information

TCB(pkg_index,info_reqd[,ERR=stmtref])

The following two codes are used in **TCB()** to report ProvideX licensing information in numeric form (owner code, package ID, and date):

pkg_index A value of 1 returns information for the first add-on package installed, a 2 indicates the second, a 3 indicates the third, and so on (20 max). A 0 *zero* returns information for the base system.

info_reqd Indicates which information will be reported; i.e.,

- 0 **Package ID** (20005=Smart Lists, 20012=Report Writer, etc.). 0 *zero* represents the base system. Refer to the website www.pvx.com for a list of valid add-on package IDs.
- 1 **Expiry Date** in numeric form YYYYMMDD. 0 means *no expiry date*.
- 2 **Activation Flags**. 32-bit numeric - for internal use.
- 3 **Users**. Total number of users per package.
- 4 **Slots**. Number of available user slots.

Example:

```
0010 FOR ID=0 TO 100
0020 PACKAGE=TCB( ID, 0, ERR=*BREAK )
0030 EXPIRY=TCB( ID, 1 )
0040 FLAGS=TCB( ID, 2 )
0050 PRINT "PACKAGE#: ", TBL( PACKAGE=0, STR( PACKAGE ), "PROVIDEX" ),
0060 PRINT @( 20 ), "EXPIRES: ", TBL( EXPIRY=0, STR( EXPIRY ), "<NEVER>" ),
0070 PRINT @( 40 ), "ACT FLAGS: ", "$"+HTA( BIN( FLAGS, 4 ) )+"$"
0080 NEXT ID
```

For more on the use of **TCB()** for reviewing activation status, see *Troubleshooting* in the *ProvideX Installation Guide*.

Format 3: *Return Platform/User Information*

TCB(keyword [,arg] [,ERR=stmtref])

Various information can be retrieved from the OS by issuing specific *keyword* values with the **TCB()** function.

The following keywords are used alone (without an argument) to return information about the platform that ProvideX is compiled for or on:

"compiled_bits"	Either 32 or 64.
"compiled_cpu"	The type of processor (x86/PowerPC etc.)
"compiled_architecture"	CISC/RISC/EPIC (nature of the CPU)
"compiled_os"	OS name
"compiled_osver"	Operating System Version

These keywords are used together with a user ID *arg* to return information associated with a specific user in a UNIX/Linux environment:

"OS_GetUserUID"	User ID.
"OS_GetUserGID"	Group ID
"OS_GetUserName"	User name.
"OS_GetUserHome"	Home directory
"OS_GetUserShell"	Shell directory.

The above keywords are only valid under UNIX and Linux and will currently return null in a Windows environment.

Examples:

```
PRINT TCB("OS_GetUserUID", "johndoe")
500
PRINT TCB("OS_GetUserGID", "johndoe")
501
PRINT TCB("OS_GetUserName", "johndoe")
Johnathan Doe
PRINT TCB("OS_GetUserHome", "johndoe")
/home/johndoe
PRINT TCB("OS_GetUserShell", "johndoe")
/bin/bash
```

TMR() Function

Timer

Format **TMR(*tmr_code* [,ERR=*stmtref*])**

Where:

tmr_code Value indicating what type of information to be returned. Numeric expression, integer range 0 to 2. See the description below.

stmtref Program line number or label to transfer control to.

Returns Number of seconds or hours or, if reset, 0 *zero*.

Description The **TMR()** function can initialize the timer (to zero) and return the time difference in seconds or hours since the last initialization. The **TMR()** function is used as follows:

TMR(0) Returns the difference in seconds since the last **TMR(0)** and *resets* the timer to 0.

TMR(1) Returns the difference in seconds since the last **TMR(0)**.

TMR(2) Returns the difference in hours since the last **TMR(0)**.

Example

```

00010 print 'CS'
00015 a=tmr(0)
00020 button 10,@(1,1,20,2)="TMR System Function"
00030 while 1
00040 obtain x
00050 if ctl=10 then gosub ShowTime
00060 if ctl=4 then break
00070 wend
00080 end
00090 ShowTime:
00100 print @(1,20),"Number of seconds since last occurrence of TMR(0):",tmr(1)
00110 print @(1,21),"Number of hours since last occurrence of TMR(0):",tmr(2)
00120 a=tmr(0)
00130 return

```

TRX() Function

Convert Radix-40 to ASCII

Format **TRX(*rdx_40\$* [, *ERR=stmtref*])**

Where:

rdx_40\$ Character string containing Radix-40 data.

stmtref Program line number or label to transfer control to.

Returns ASCII equivalent of given Radix-40 data.

Description The **TRX()** function converts data from Radix-40 to normal ASCII. The input to this function is generally the result of a call to the **RDX ()** function, which performs the inverse of converting ASCII to Radix-40. Radix-40 lets you compress three characters into two bytes of data (16 bits).

The compression algorithm only supports the characters A-Z, 0-9, ., -, \$ or space. Lower case letters are converted to their uppercase equivalent. All other characters are replaced with spaces.

See Also [RDX\(\) Convert ASCII to Radix-40, p.509.](#)

Examples

```
0010 LET A$=LST(PGM(65000)); PRINT "string: ",QUO,A$,QUO
0020 LET B$=HTA(RDX(A$)); PRINT B$
0030 PRINT HTA(TRX(B$))
-:run
string: "65000 END "
2CB106686E600000
37304F394B4D36504A374F20374F41374E54365044365044
```

TSK() Function

Returns Entry from Task List

Format **TSK(num[,ERR=stmtrf])**

Where:

num Desired index entry in the internal task table. Numeric expression, range 0 to *n*.

stmtrf Program line number or label to transfer control to.

Returns String, value of task internal task table.

Description The **TSK()** function returns a string value from the internal task table. The returned table entry is determined by the task number specified, *num*. An error is generated if the entry has not been set up in the task table (using a **SETDEV TSK()** directive).

See Also **SETDEV TSK() Directive, p.314.**

Examples

```
0010 SETDEV TSK() "this is table entry 0"
0020 SETDEV TSK() "this is table entry 1"
0030 PRINT TSK(1)
->run
this is table entry 1
```



Note: An argument of -1 always returns " [Pvx " which is the internal header for files.

TXH() Function

Text Height

Format TXH(*string\$* [, *chan*] [, *ctrlopt*])

Where:

chan Channel or logical file number of the device.

ctrlopt Control options. Supported options for TXH() include:
ERR=stmtref Error transfer
FNT="font",size[,attr]" Font name, size, properties
SIZ=num

string\$ String expression.

Returns Text height in graphical units (i.e., @X(*col*) and @Y(*line*) values).



Note: For use in graphics mode along with [TXW\(\) Text Width, p.545](#).

Description The TXH() function returns text height in graphical units for the given *string\$*. By default, if no **SIZ=** is specified, this function returns the height of a single line of text in either the font specified or the current font on the device specified by *chan*.

If you set a size, TXH() returns the height required to print your given text with no line exceeding the size (with word wrap enabled). The **SIZ=** option thus can be used to determine the height of the text.

If the font selected indicates that the ampersand character is used to denote underscored characters, TXH() does not include & during its calculations.

If *chan* is omitted, ProvideX assumes it is dealing with the main window.

Example

```
0900 PRINT 'CS'
1000 LET R$="Hello World"
1001 LET W=TXW(R$) ! Get width of text
1002 IF W>@X(40) THEN LET W=@X(40) ! Force <= 40 columns
1010 LET H=TXH(R$,SIZ=W) ! Make it fit in width
1020 PRINT 'TEXT'(@X(20),@Y(5),@X(20)+W,@Y(5)+H,R$,"W")
```


TXW() Function

Text Width

Format TXW(*string*[\$[,*chan*][, *ctrlopt*])

Where:

chan Channel or logical file number of the device.

ctrlopt Supported options for TXW() include:

ERR=stmtrf Error transfer

FNT="font,size[,attr]" Font name, size, properties

SIZ="wrapwidth" Word wrap to maximum width

string String expression.

Returns Text width in graphical units (i.e., @X(*col*) and @Y(*line*) values).



Note: For use in graphics mode along with TXH() [Text Height](#), p.544.

Description The TXW() function returns text width in graphical units for the given *string*\$. If *chan* is omitted, ProvideX assumes it is dealing with the main window.

If the font selected indicates that the ampersand character is used to denote underscored characters, TXW() does not include & during its calculations.

Example If you want to create a window based on the length of a message, use TXW() to determine the size of the window /dialogue to create; e.g.,

```
0055 LET LINE1$="Line 1."
0056 LET LINE2$="Line 2.."
0057 LET LINE3$="This is Line 3..."
0100 LET W1=TXW(LINE1$,FNT="MS Sans Serif")
0110 LET W2=TXW(LINE2$,FNT="MS Sans Serif")
0120 LET W3=TXW(LINE3$,FNT="MS Sans Serif")
0130 LET W=MAX(W1,W2,W3)
0140 LET W=INT(W/16)+4 ! Allow for up to 4 extra white spaces
0150 PRINT 'DIALOGUE'(C,L,W,5,TITLE$),'SR','CS',
0160 PRINT 'FONT'("MS Sans Serif"),
0170 PRINT 'TEXT'(@X(2),@Y(1),LINE1$),
0180 PRINT 'TEXT'(@X(2),@Y(2),LINE2$),
0190 PRINT 'TEXT'(@X(2),@Y(3),LINE3$),
0200 MSGBOX "Press OK to continue."
```



Note: The logical width is always 16 times the column number. In the 'TEXT' mnemonic, if you omit end points, the system will calculate them.

UCS() Function

Return Upper Case String

Format UCS(string\$[,ERR=stmtref])

Where:

stmtref Program line number or label to transfer control to.

string\$ String expression whose upper case ASCII counterpart is to be returned.

Returns Uppercase string equivalent of lower case string.

Description The **UCS()** function replaces all lower case ASCII characters in a given string with their corresponding upper case values.

See Also [LCS\(\) Return Lowercase String, p.472](#)
[DEF systab= Directives, p.74](#)

Example

```
0010 INPUT "Enter name: ",NAME$
0020 LET NAME$(2)=LCS(NAME$(2))
0030 LET NAME$(1,1)=UCS(NAME$(1,1))
0040 PRINT "Name is: ",NAME$
-:run
Enter name: rOBERT
Name is: Robert
```



Note: If an *asterisk* * is passed to the **UCS()** function instead of a string, the function returns the 256-byte Uppercase Conversion Table.

UCP() Function

UnCompress Data

Format UCP(*string*[,ERR=*stmtref*])

Where:

string Original data to expand (uncompress).

stmtref Program line number or label to transfer control to.

Returns Expanded data string.

Description The **UCP()** function expands a string that has been compressed using the **CMP()** function. For a usage example, refer to the **CMP() Function, p.404**. **TCB(195)** will return 1 if ZLib support is available.

Interfacing with Other ZLib-compliant Utilities

The data returned from the **CMP()** function includes a single header byte (value between \$01\$ and \$FF\$) to facilitate ZLib uncompression routines. The **UCP()** requires this value to expand the data.

If data has been compressed using ZLib utilities that are outside of ProvideX, this value will not be present; therefore, a \$00\$ header must be inserted in its place. This will cause the **UCP()** function to attempt to uncompress multiple times into varying buffer sizes until the data can be uncompressed.

If the buffer size required for the uncompressed data is known, the header byte can be calculated and used; i.e., if you know that the compressed data is less than a maximum of 10,000 bytes long, and the actual length is 2,000, you can prefix the data with \$05\$ indicating that the output buffer should be set to 5 times the compressed data size. Should a header value be given, but the data still doesn't fit, the system will fall back into multiple uncompress attempts while increasing the output buffer until it is successful.



Note: Since the **CMP()** and **UCP()** compression routines are not supported on all platforms, systems using these functions may not be fully portable.

See Also **PCK() Function, p.498**
 CMP() Function, p.404

UPK() Function

Unpack Numeric Data

Format **UPK**(string\$,ERR=stmtref)

Where:

stmtref Program line number or label to transfer control to.

string\$ String expression whose value represents a packed number.

Returns Numeric expression of whose value has been packed into a string.

Description **UPK()** is used to convert a packed string into its numeric value. It is the counterpart of the **PCK()** function.

The packing algorithm used takes a numeric value and splits it into a series of two digit values where each of the two digit values represents a number between 0 and 99. These numbers are then added to 32 to create the series of single-byte printable characters that comprise the packed string. To unpack the value each byte of the string has 32 subtracted from it and the resultant values become a series of 2-digit values in the final result.

Should the value of any two-digit pair (when added to 32) equal or exceed the standard file separator (\$8A\$), the value will be incremented by one when the output string is created. When unpacking the string, any byte exceeding the field separator will be reduced by one prior to subtracting 32.



Note: This function is not necessarily compatible with all Business Basics

See Also

PCK() Pack Numeric Data, p.498

CMP() Compress Data, p.404

VIN() / VIS() Functions Obtain Value of Variable

Formats

1. *Get Numeric Value*: **VIN**(string\$[,ERR=stmtref])
2. *Get Numeric From Composite*: **VIN**(composite\$,var_1\$[[,subscr]]=expr[, ... n])
3. *Get String Value*: **VIS**(string\$[,ERR=stmtref])
4. *Get String From Composite*: **VIS**(composite\$,var_1\$[[,subscr]]=expr[, ... n])
5. *VIS() as Record Prefix*: **REC=VIS**(string\$[,ERR=stmtref])

Where:

<i>composite\$</i>	String variable which has been defined as a composite string.
<i>string</i>	String expression containing the name of a variable to be used.
<i>stmtref</i>	Program line number or label to transfer control to.
[,subscr]	Optional subscript(s) of a variable in the composite string. String expression. You can include from 1 to 3 optional numeric expressions in square brackets, comma-separated, as subscripts for the variable.
<i>var_1\$</i> to <i>var_2\$</i>	String expression containing the name of a variable in the composite string.



Note: You can use the optional **ERR=stmtref** with each of the syntax formats described. The inner set of brackets enclosing [,subscr] are part of the syntax.

Returns

Contents (value) of variable(s), numeric or string.

Description

The **VIN()** function returns the numeric value (contents) of a variable where the name of the numeric variable is stored in a string variable. The **VIS()** function returns the string value (contents) of a string variable whose name is stored in a string variable.

Format 1: Get Numeric Value

VIN(string\$[,ERR=stmtref])

The **VIN()** function returns the numeric contents of a numeric variable whose variable name is stored in *string\$*.

Format 2: Get Numeric From Composite

VIN(composite\$,var_1\$[[,subscr]]=expr[, ... n])

The **VIN()** function returns the value contained in each numeric variable you name (where the variable is part of a composite string). You can choose to specify up to three subscripts for the variable.

Format 3: Get String Value**VIS(string\$[,ERR=stmtref])**

The **VIS()** function returns the contents of a string variable whose variable name is stored in *string\$*.

Format 4: Get String Value From Composite**VIS(composite\$,var_1\$[,subscr]=expr[, ... n])**

The **VIS()** function returns the value contained in each string variable you name (where the variable is part of a composite string). You can choose to specify up to three subscripts for the variable.

Format 5: VIS() as Record Prefix**REC=VIS(string\$[,ERR=stmtref])**

Use this format to assign a **REC=** prefix value dynamically during **READ**, **WRITE**, **OPEN**, and other directives. When **REC=VIS()** is used, ProvideX evaluates the string value in **VIS()** and uses it as the name of the record prefix.

Example

```
00010 print 'CS'
00020 print "Name$,StreetNum,Street$,City$"
00030 Name$="Mike",StreetNum=8920,Street$="Woodbine Ave.",City$="Markham"
00040 input "Select field type (N=Numeric,S=String)",T$
00050 if T$="" then stop
00060 if ucs(T$)<>"S" and ucs(T$)<>"N" then goto 0030
00070 input "What field?",F$
00080 if F$="" then stop
00090 if ucs(T$)="S" then print F$,"=",vis(F$) else print F$,"=",vin(F$)
00100 goto 0030
```

XEQ() Function

In-line Subprogram Execute

Format XEQ(subprog\$,expression,arg1,arg2,...[,ERR=stmtref])

Where:

arg1,arg2... Comma-separated list of arguments to pass to the subprogram.

expression Expression will be evaluated after the call and used as the return value for the function.

subprog\$ Name of the subprogram to call. Maximum string size 8kb.

stmtref Program line number or label to transfer control to.

Returns Evaluated value in expression after executing in-line **CALL**.

Description The **XEQ()** function executes an in-line **CALL** directive. When ProvideX encounters the **XEQ()** function, the subprogram named in the function will be called with any arguments supplied. When the subprogram exits, the **XEQ()** function evaluates the expression and returns it.

The primary advantage of **XEQ()** is in single-line Global functions. Consider the following:

Was:

```
0010 CALL "GETDTE" ,DT,DATE$
0020 PRINT "Date:" ,DATE$
```

Now:

```
0010 PRINT "Date:" ,XEQ("GETDTE" ,_D$ ,DT ,_D$)
```

Or, you could define a Global function in a startup program:

```
DEF FN%DTE$(_DT)=XEQ("GETDTE" ,_D$ ,_DT ,_D$)
```

Then:

```
0010 PRINT "Date:" ,FN%DTE$(DT)
```

XFA() Function

Extended Field Attributes

Format XFA(*varlist* [, *ERR=stmtref*])

Where:

stmtref Program line number or label to transfer control to.

varlist List of variables in the string template.

Returns Extended field-attribute information in string template.



Note: This function is included for compatibility with other languages.

Description The XFA() function returns extended field attribute information stored in a string template:

Format	Returns
XFA(<i>var</i> , " ")	A list of variables.
XFA(<i>var</i> , " <i>fieldname</i> ")	Description of the field.
XFA(<i>var</i> , " <i>fieldname</i> ", " <i>userfield</i> ")	User-defined portion of the field.

Do not do this:

```

0120 DIM CST$:IOL=0130 REM does NOT create a valid string template for XFA( )
0130 IOL=NAME$,ADDR$,CITY$,ZIP$ REM valid IOL for the composite string above
0140 REM But XFA(string above) generates Error #26: Variable type invalid
0150 REM

```

Apply this instead:

```

0170 REM XFA( ) is for use with other Business Basics' string template formats
0180 LET J$="B JONES",K$="23 SOME ST.",L$="MYCITY",M=78923
0190 DIM CST$:"NAME:C(20*),ADDR:C(30*),CITY:C(20*),ZIP:N(10*):type=cur"
0200 LET CST.NAME$=J$
0210 LET CST.CITY$=K$
0220 LET CST.ADDR$=L$
0230 LET CST.ZIP=M
0240 REM
0250 PRINT XFA(CST$, " ")
0260 PRINT XFA(CST$, "NAME")
0270 PRINT XFA(CST$, "ZIP", "type")
--:END
--:RUN

```

```

NAME
ADDR
CITY
ZIP

```


`_Ä``cur`

```
-> HTA(XFA(CST$, "NAME" ))
01C00A0001000100000014
```

The hex string of the **XFA()** function in the previous example reports the following information about the string template field attributes.

<i>Bytes</i>	<i>Information Returned</i>
1,1	Code 1-8 for field type
2,1	Flag bit
3,1	Field terminator
4,2	Size of repeating field or \$0001\$ if not repeating
6,2	Field number, based on line feeds.
8,2	Field offset if prior field is variable length
10,2	Field length
12,	User defined attributes

XOR() Function

Logical Exclusive OR

Format XOR(value1[\$],value2[\$][,ERR=stmtref])

Where:

stmtref Program line number or label to transfer control to.

value1[\$] Compared values. String or numeric expressions/variables. If strings,
value2[\$] *value1\$* must be the same length as *value2\$*

Returns Result of logical exclusive 'OR' comparison of two expressions/variables.

Description The **XOR()** function performs a bit-wise exclusive 'OR' comparison of two string or numeric expressions/variables, and generates a value as a result. The length of the two string expressions must be equal or ProvideX returns an Error #46: Length of string invalid.

<i>Binary</i>	<i>Result</i>
0 XOR 0	0
1 XOR 0	1
0 XOR 1	1
1 XOR 1	0

Sample Comparison Results:

XOR(\$41,\$42) yields Hex 03, 00000011

XOR(\$41,\$25) yields Hex 64, 01100100

XOR(\$5A,\$DD) yields Hex 87, 10000111

See Also [IOR\(\) OR Comparison, p.460](#)
[AND\(\) Logical AND, p.394](#)

Examples

```
0040 READ (1,END=1000)F$
0050 R$=XOR(F$(1,2),$2020$) ! Convert to lower case
0060 ...
```

```
IF POS($00$=XOR(UCS(X$),LCS(X$)))=0 THEN PRINT X$," is all alpha!"
```



4

System Variables

BKG	GID	NAR	SID
CHN	HFN	NID	SSN
CTL	HLP	PFX	SYS
DAY	HWD	PGN	TIM
DLM	LFA	PRC	TME
DSZ	LFO	PRM	TMS
EOM	LIP	PSZ	TSM
ERR	LPG	QUO	UID
ERS	LWD	RET	UNT
ESC	MSE	RND	WHO
GFN	MSL	SEP	

Overview

This chapter provides an alphabetically arranged list of all the system variables in ProvideX. Each definition includes the correct syntax (showing associated parameters), values returned, a general description, examples, and sometimes a cross reference to related documentation. The list begins on the following page.



Note: All system variables have *reserved three-character names*. To avoid potential conflicts with the reserved list (since ProvideX might include more three-character variables in the reserved list in future) it is best not to use three-character variable names in your applications.

BKG System Variable

Background Process Status

Numeric System Variable

Contents Integer, process status code.

Description The **BKG** variable contains the following numeric status codes:

- 0 *Zero.* Current program is directly connected to a terminal user (background/ghost process).
- 1 Current program is not directly connected to a terminal user (background/ghost process).

Examples `->?BKG`
0

CHN System Variable

Channels Open

Numeric System Variable

Contents Numeric string, current open channels.

Description This system variable contains a listing of all currently open channels. The value returned is a string of numeric two-byte values. The **CHN** variable indicates channel values below 65000.

Examples To obtain the hex values, use the **HTA()** function.

```
->?HTA(CHN)
0000001E
```

In the above example, `DEC(0000)=0` (i.e., the console) and `DEC($001E$)=30` (the open file's channel).

```
->?(LEN(CHN)/2)-1
```

Yields the number of files opened.






CTL System Variable

Control Signal Code

Numeric System Variable

Contents Integer, control signal code.

Description The **CTL** system variable contains a numeric code (integer) that represents a signal of user input from the keyboard or mouse. From the keyboard, this code represents how the user ended the last **INPUT** statement; i.e.,

0	 key (normal)
1 - 4	 to  keys
5	Input terminated due to SIZ= option
6 - 12	 to  keys

Positive **CTL** codes (in the range 0 to 32767) represent function keys as well as user-defined control signals that are to be returned to the application. Negative **CTL** codes are handled internally by ProvideX – see [Negative CTL Definitions, p.817](#).

See Also [DEFCTL Define/Redefine CTL Values, p.78](#)
[CTL\(\) Return CTL Definition, p.410](#)
[SETCTL GOSUB on CTL Event, p.307](#)
[DEF sysvar= Define System Variables, p.76](#)

DAY System Variable

Return Current System Date

String System Variable

Contents String, current system date, formatted.

Description The **DAY** variable contains the current system date (e.g., 11/15/00), in a format based on the date style set in the **DAY_FORMAT** directive (MM/DD/YY by default). Changes in **DAY_FORMAT** are reflected in the value returned in the **DAY** variable.

See Also [DAY_FORMAT Directive, p.64](#).

Examples

```
0100 PRINT DAY, " ",
0110 DAY_FORMAT "DD/MM/YYYY"; PRINT DAY
-:run
11/15/99 15/11/2000
```

DLM System Variable Return System Directory Delimiter

String System Variable

Contents String, operating system's directory delimiter.

Description This system variable contains the operating system's directory delimiter (e.g., "/" ... for UNIX, "\" ... for Windows). For example, on a Windows PC:

```
->?DLM
\  

```



Tip: Use SEP=DLM when reading directories (e.g., in list boxes) to have ProvideX append the operating system delimiter to subdirectory names.

Examples The following is a tree view list box example.

```
0010 LIST_BOX 10,@(5,5,25,10),OPT="e|",
0010:FMT="{!diskette|!File|!File_open}",SEP=DLM
0020 LET F=1,D$="."
0030 NXTDIR:
0040 OPEN (F)D$
0050 NXTFILE:
0050:READ (F,END=ENDDIR)F$
0060 IF F$(1,1)="."
0060:THEN GOTO NXTFILE
0070 IF MID(F$,-1)<>DLM
0070:THEN LIST_BOX LOAD 10,0,PTH(F)+DLM+F$;
0070:GOTO NXTFILE
0080 LET D$=PTH(F)+DLM+F$
0090 F++
0100 GOTO NXTDIR
0110 ENDDIR:
0110:CLOSE (F)
0120 F--
0130 IF F>0
0130:THEN GOTO NXTFILE
0140 ESCAPE
```

DSZ System Variable *Data Space Size Available to User*

Numeric System Variable

Contents Integer, amount of data work space available to the user.

Description The **DSZ** variable contains a numeric value that indicates the size (in bytes) of the available data work space for the user. You can set the size value either in the **START** directive or as an argument in the ProvideX command (using **-SZ**).

Examples
->?DSZ
291094

EOM System Variable *End of Message Character String*

String System Variable

Contents String, End-Of-Message character.

Description This system variable contains the End-Of-Message character string that ended the user's last input from the terminal. The contents of this variable will vary based on the type of terminal being used.

See Also [DEF sysvar= Define System Variables, p.76](#)

Examples The carriage return (not printable) is the EOM:
->?HTA (EOM)
0D

ERR System Variable Last System-Detected Error Value

Numeric System Variable

Contents Integer, last system-detected error code.

Description The **ERR** variable contains a numeric value (integer) that indicates the last system-detected error. It is reset by the **BEGIN**, **CLEAR**, **END**, **RESET**, **STOP**, and **START** directives. See [Error Codes and Messages, p.828](#), for a complete list of codes.

Examples In the following example, the "TEST" file is already open:

```
0100 OPEN (5)"TEST"
Error #14: Invalid I/O request for file state
->?ERR
14
```

See Also [ERR\(\) Function, p.427](#)
[DEF sysvar= Define System Variables, p.76](#)

ERS System Variable Line Number of Last Error

Numeric System Variable

Contents Integer, line number that generated last error.

Description This **ERS** variable contains the line number that generated the last error detected in the program.

Examples

```
0030?LET X$="" PRINT "Reset"
Error #20: Syntax error
1>RUN
0030?Let X$="" print "Rest"
Error #20: Syntax error
->?ERS
30
> EDIT 30
```


ESC System Variable

ASCII ESCape Character

String System Variable

Contents String, ASCII escape character.

Description This system variable contains the ASCII escape character `$1B$`.

Examples

```
->?HTA(ESC)
1B
->
```

GFN System Variable

Highest Available Global Channel

Numeric System Variable

Contents Integer, highest available global channel.

Description The **GFN** variable contains a numeric value (integer) representing the highest available (i.e., not open) global file channel. This value will be in the range: 64 to 127 (32768 to 65000 if the '**XF**' system parameter is set).

Examples

```
->set_param 'xf'
->?gfn
65000
->SET_PARAM - 'XF'
->?GFN
127
```

GID System Variable *Operating System Process Identifier*

String System Variable

Contents String, two-character OS process identifier.

Description This system variable can be used to obtain a two-character 16-bit binary operating system group or Process ID (PID).

Examples Obtain the 16-bit UNIX Process ID by using:

```
->pid=dec($00$+gid)
->?pid
30633
```

For larger values (e.g., 32-bit), refer to `TCB(89)` under the [TCB\(\) Function, p.534](#). `TCB(89)` returns the numeric value of the current PID. Use this as a replacement for `DEC(00+GID)` to retrieve the current PID value.

See Also [SYS\(\) Function, p.529](#)

HFN System Variable *Highest Available Local Channel*

Numeric System Variable

Contents Integer, highest local channel/file number not open.

Description The **HFN** variable contains a numeric value (integer) representing the highest available local channel/file number. This value will be in the range: 0 to 63 (0 to 32767, if the 'XF' system parameter is set).

Examples

```
->?HFN
63
->SET_PARAM - 'XF'
->?HFN
32767
```

HLP System Variable *Last Specified* HLP= Value

String System Variable

Contents String, HLP= value from **INPUT** or **OBTAIN** directive.

Description This system variable contains the HLP= value specified in the last **INPUT** or **OBTAIN** directive. If the HLP= option is omitted, this variable contains a null string.

HWD System Variable *Starting/Home Directory*

String System Variable

Contents String, home directory name.

Description The **HWD** variable contains the name of the directory that was current at start up (when ProvideX was initialized). This is considered the *home* directory.

Examples
->?HWD
C:\Program Files\Sage Software\ProvideX

LFA System Variable *Last File Number Accessed*

Numeric System Variable

Contents Integer, channel/file number of last device/file accessed.

Description This system variable indicates the channel of the last file or device accessed. The value is 0 *zero* if the last device was the console/user's terminal.

Examples
->?LFA
0

See Also [DEF sysvar= Define System Variables, p.76](#)

LFO System Variable

Last File Number Opened

Numeric System Variable

Contents	Integer, channel/file number of last file opened.
Description	This system variable indicates the channel/file number of the last file opened.
Examples	0100 OPEN (HFN) "TESTFILE" 0110 CHANNEL=LFO
See Also	DEF sysvar= Define System Variables, p.76

LIP System Variable

Input Location: Column, Line

String System Variable

Contents	Numeric string, screen line and column location of last input.
Description	The LIP variable indicates the screen position (i.e., the column and line numbers at the intersection) where the last user input occurred. This information is used by the Help/Query subsystem.
Examples	->?LIP 0212

LPG System Variable

Lead Program Name

String System Variable

Contents	String, lead program name.
Description	This system variable contains the name of the lead program; i.e., the program that started the current ProvideX session or the last program specified in the START directive.
See Also	ProvideX Installation Guide, Launching ProvideX
Examples	->?LPG start.win

LWD System Variable

Current Working Directory

String System Variable

Contents String, pathname of current working directory

Description This system variable contains the full pathname of the current working directory.

Examples

```
->?LWD
C:\Program Files\Sage Software\ProvideX
->
```

MSE System Variable

Mouse State

String System Variable

Contents String, current state of mouse.

Description This system variable contains a 32-byte string that describes the current state of the mouse. The following table shows the positions of all mouse attributes represented in the **MSE** string:

<i>MSE String</i>	<i>Description</i>
(1,1)	Current state: \$FF\$ Mouse inactive/unavailable. \$00\$ Character-based environment, no mouse. \$01\$ Left button down. \$02\$ Right button down. \$03\$ Both buttons down.
(2,1)	Current mouse column in binary.
(3,1)	Current mouse line in binary.
(4,2)	Current mouse X (column) position in binary.
(6,2)	Current mouse Y (line) position in binary.
(8,1)	Absolute column #.
(9,1)	Absolute row #.
(10,1)	Standard character width.
(11,1)	Standard character height.
(12,1)	Width of Scroll box on standard scroll bar.
(13,1)	Height of Scroll box on standard scrollbar.
(14,2)	Mouse X (column) position relative to current window in binary.

<i>MSE String</i>	<i>Description</i>
(16,2)	Mouse Y (line) position relative to current window in binary.
(18,2)	Current control object with focus.
(20,2)	Last control object with focus, relative to the current window.
(22,1)	WindX/JavX revision level from \$00\$ to \$FF\$ (\$FF\$ for no thin-client).
(23,2)	CTL value for context-sensitive help.
(25,2)	Last control object to lose focus, relative to the current window.
(27,2)	Maximum X (column) coordinates of the screen.
(29,2)	Maximum Y (line) coordinates of the screen.
(31,1)	Returns current window state. Values correspond to 'SHOW' () mnemonic, i.e., <code>dec(mid(mse, 31, 1)) =</code> -1 for hidden 0 for minimized 1 for normal 2 for maximized.
(32,1)	Returns either w for WindX or J for JavX (\$00\$ for no thin client).

Examples

```
IF MID(MSE, 22, 1) > $00$ AND MID(MSE, 22, 1) < $FF$ THEN %WDX$ = "[WDX]"
```

To get the maximum screen size in lines and columns:

```
0010 LET X$=MSE
0020 XCHAR=DEC($00$+X$(10,1))
0030 YCHAR=DEC($00$+X$(11,1))
0040 MAX_WIDE=INT(DEC(X$(27,2))/XCHAR)
0050 MAX_HIGH=INT(DEC(X$(29,2))/YCHAR)
```

MSL System Variable *Length of String Matching Last MSK*

String System Variable

Contents	Integer, length of string matching last MSK() function.
Description	The MSL variable contains a numeric value (integer) that indicates the length of the string that matched the last MSK() function. TCB(16) also reports the length of the string.
See Also	MSK() Function, p.486 , TCB() Function, p.534 .
Example	<pre>-->PRINT MSK("my name is Foxy", "[A-Z][a-z]*"),MSL 12 5</pre>

NAR System Variable *Number of Arguments, Start ProvideX*

Numeric System Variable

Contents	Integer, number of arguments in ProvideX start up.
Description	This system variable contains a numeric value (integer) representing the number of arguments in the ProvideX command that launches ProvideX (e.g., in using a batch file or from a command statement).
See Also	ARG() Function, p.395 , <i>ProvideX Installation Guide, Launching ProvideX</i>
Example	<pre>--> PVX -SZ=20 -ARG TOM JONES ->?NAR 2</pre>

NID System Variable *Network or Network Node ID*

String System Variable

Contents	String, network identifier.
Description	The NID variable contains the network identifier. Under UNIX, this variable contains the network node name.

PFX System Variable

Current Prefix Setting


String System Variable

Contents	String, current pathname prefix for PREFIX (0).
Description	This system variable indicates the current settings of the PREFIX directive (for PREFIX entry 0).
See Also	PREFIX Directive, p.249.
Example	<pre>->?PFX \PGMS\===\ \PGMS\====\ \PGMS\CUST\CASHRCPT\ \PGMS\CUST\MSC\ \PGMS\CUST\SALES\</pre>

PGN System Variable

Current Program Pathname

String System Variable

Contents	String, name of currently loaded program.
Description	This system variable contains the name of the currently loaded program, complete with its full operating system pathname.
	<p> Note: There is one exception to the above. If you have the 'OP' (original program) system parameter set to ON, PGN returns only the program name (i.e., without an expanded pathname).</p>

See Also ['OP' System Parameter, p.678.](#)

Example	<pre>0100 PRINT PGN 0110 SET_PARAM 'OP' 0120 PRINT PGN -:run C:\Program Files\Sage Software\ProvideX\TST\TST_EGS TST_EGS</pre>
---------	--

PRC System Variable Precision Currently In Effect

Numeric System Variable

Contents Integer, current **PRECISION**.

Description This system variable contains a numeric value (integer) that indicates the current **PRECISION** in effect (except in scientific notation). This value will be in the range: -1 to 18. The default is 2 (two digits to the right of the decimal point).

If the mode is **FLOATING POINT**, then the value returned is in scientific notation. Either a **PRECISION -1** statement or a **FLOATING POINT** directive will activate scientific notation. The **PRECISION** directive cancels it.

See Also ['PD'= System Parameter, p.679](#)
[PRECISION Directive, p.248](#)
[PRC\(\) Function, p.503](#)

Examples

```
0010 PRECISION 14; FLOATING POINT ; LET A=7.1234
0020 PRINT PRC,@(10),A
0030 BEGIN ! Resets A=0, precision=2 (cancels scientific notation)
0040 PRINT PRC,@(10),A
0050 LET A=6.1234
0060 PRINT PRC,@(10),A
-:run
.14E+02   .71234E+01
2   0
2   6.12
```

PRM System Variable ProvideX Parameter Settings

String System Variable

Contents	String, comma-delimited list of current system parameter settings.
Description	This system variable contains a comma-delimited string listing the current settings of OS parameters for ProvideX use. See also Chapter 6. System Parameters, p.653 . Some parameters only appear in the list when set, and some are OS specific.

Example

```
PRINT PRM ! Version 4.20 WINDOWS settings
-'3D',-'AD',-'AH', 'AI'=10,-'B0', 'BF'=10,-'BT',-'BX', 'BY'=1970,-'CD', 'CS', 'CT'=0,
'CU'=36,-'D0', 'DC', 'DF'=0, 'DL'=0, 'DP'=46, 'DT'=0, 'DW'=0,-'EG', 'EL'=0,-'EO',-'ES',
-'EX',-'F4', 'FB'=5,-'FC', 'FF'=0,-'FI', 'FO'=0,-'FU',-'FL', 'FP', 'FS'=138,-'FT',-'F
X',-'F',-',-'I0',-'I2', 'IC',-'IM', 'IR', 'IS'=5,-'IZ',-'KR', 'LB'=4,-'LC',-'LD',-'LE'
, 'LS'=1,-'LU',-'LZ', 'MB'=0, 'MF'=50,-'MP', 'NE',-'NI',-'NK',-'NL',-'NN',-'NR',-'OC
', 'OL'=25, 'OM',-'OP', 'OR', 'OW'=0, 'PC'=10, 'PD'=2,-'PO',-'PU', 'PW'=36,-'PZ', 'QF'=1
, 'Q_ '=2, 'Q^ '=2,-'QS',-'QT',-'RI', 'RN'=1, 'RP',-'RR',-'RS',-'SC',-'SD',-'SF',-'SK'
, 'SL'=32,-'SP',-'SR', 'SV'=1, 'SZ'=32000,-'TB', 'TC'=0, 'TH'=44,-'TL',-'TN',-'TT',-'
TU',-'TX', 'VP'=48, 'VR'=0, 'VW'=0, 'WB', 'WD'=10000,-'WF', 'WH'=0, '
'=2,-'XC',-'XF',-'XI',-'XT',-'ZP',-'DD', '!B'=3, '!U'=0,-'IU'
```

You can have the **PRM()** function return a specific parameter's current setting (or the Boolean value, 1=ON / 0=OFF, for a switch), even when it's hidden from the PRM listing.

```
->?prm('!i') ! hidden unless ON
0
```

PSZ System Variable Current Program Size

Numeric System Variable

Contents	Integer, program size
Description	This numeric system variable indicates the current program size in bytes.

Example

```
->?PSZ
2605
```

QUO System Variable

ASCII Quote Character

String System Variable

Contents String, ASCII quote (") character.

Description This string system variable contains the ASCII quote character (").

Example `COMMAND$="SAVE "+QUO+PGN+QUO`

RET System Variable

Operating System's Last Error Code

Numeric System Variable

Contents Integer, operating system error + 256.

Description This numeric system variable returns an integer reporting the operating system's error code associated with the last operating call. ProvideX generates this value by adding 256 to the error code value in the **ERR** system variable.

Example

```
-: PRINT "RET = ",RET
RET = 258
```

See Also [ERR Last System-Detected Error Value, p.560](#)
[DEF sysvar= Define System Variables, p.76](#)

RND System Variable

Random Number Generator

Numeric System Variable

Contents Decimal numeric, **PRECISION** 8, random number.

Description This system variable contains a different random number each time you use it to return a value. The value will be in the range from 0 to 1, with a **PRECISION** of 8.

Example

```
0010 FOR I=1 TO 3
0020 PRINT RND,
0030 NEXT
0040 PRINT " DONE"; END
-:run
0.76559375 0.5199119 0.9505763 DONE
```

See Also [RANDOMIZE Directive, p.270](#)
[RND\(\) Function, p.513](#)

SEP System Variable

ProvideX Field Delimiter

String System Variable

Contents String, separator value, default \$8A\$.

Description This string system variable indicates the record field separator; the ProvideX default is \$8A\$). Note that when you **WRITE** using an IOList, **SEP** is the field delimiter.

Examples MSGBOX "Unable to save "+P\$+SEP+MSG(ERR), "Save Error"
LIST_BOX LOAD ITEMS.CTL,0, item_code\$+SEP+item_desc\$

SID System Variable

System Identification Code

String System Variable

Contents String, operating system identification code.

Description This system variable contains the system identification code as defined by the operating system.

Example ->?SID
MSDOS
->

SSN System Variable

System Software Identifier

String System Variable

Contents String, software identifier.

Description This system variable contains a sixteen character system software identifier. The first 4 characters report the version number, characters 6-8 report the machine classification, and 10-16 report the software serial number.

Example ->?SSN
0510-001-1234567
->

SYS System Variable *Operating System Identification*

String System Variable

Contents	String, name of operating system.
Description	This string system variable contains the name of the operating system.
Example	-->?SYS MS-WINDOWS

TIM System Variable *Time in Hours Past Midnight*

Numeric System Variable

Contents	Decimal numeric, PRECISION 6 , system time.
Description	This system variable returns the current system time in hours past midnight. The TME and TIM variables are identical.
See Also	SETTIME Directive, p.323
Example	0100 PRINT TIM 0110 SETTIME 1.10 0120 PRINT "Current time",TIM RUN 19.341613 Current time 1.100502

TME System Variable *Time in Hours Past Midnight*

Numeric System Variable

Contents	Decimal numeric, PRECISION 6 , system time.
Description	This system variable returns the current system time in hours past midnight. The TME and TIM variables are identical.
See Also	SETTIME Directive, p.323
Example	<pre>0100 PRINT TME 0110 SETTIME 1.10 0120 PRINT "Current time",TME RUN 19.341613 Current time 1.100502</pre>

TMS System Variable *Seconds Expired in Current Minute*

Numeric System Variable

Contents	Integer, system time in seconds passed in current minute.
Description	The TMS variable contains a numeric value (integer) indicating the current number of seconds that have passed in the current minute according to the operating system's clock. This value will be in the range: 0 to 60.
Example	<pre>->?TMS 15</pre>



TSM System Variable Error Status of Current Program

String System Variable

Contents String, error status of current program.



Note: This variable is included for compatibility with other languages.

Description This system variable indicates the error status of the currently running program. (This is included in ProvideX primarily for compatibility with other Business Basics.) Its contents are as follows:

(1,1)	"0"
(2,16)	Name of program with last error. Padded with nulls.
(18,5)	Line with error
(23,5)	Error status
(28,5)	Operating system error code

UID System Variable

Current UserID

String System Variable

Contents String, current User ID.

Description This system variable contains the current UserID. On systems without user registration, it returns the value of the environment variable USER or the Network UserID. The **UID** and **WHO** variables are identical.

Example

```

0010 ! START_UP
0020 OPEN (1) "MYCONFIG"
0030 READ (1,KEY=WHO,ERR=0050)X$
0040 SETFID X$
0050 CLOSE (1)
->?UID
SMITHJ

```

UNT System Variable *Lowest Available Local Channel*

Numeric System Variable

- Contents** Integer, lowest channel/file number not open.
- Description** The **UNT** system variable contains the lowest available channel/file number.
- See Also** [ENABLE Directive, p.110](#),
[PREFIX Directive, p.249](#).

Example

```
0010 LET CHAN_NUM=UNT
0020 OPEN (CHAN_NUM)"FILENAME"
0030 print CHAN_NUM
->?
1
```



Note: Some older Business Basics opened the printer on Channel (1). If you are dealing with such applications in legacy code, try using **HFN** instead of **UNT**. (See [HFN Highest Available Local Channel, p.562](#).)

WHO System Variable *Current UserID*

String System Variable

- Contents** String, current User ID.
- Description** This system variable contains the current UserID. On systems without user registration, it returns the value of the environment variable **USER** or the Network UserID. The **UID** and **WHO** variables are identical.

Example

```
0010 ! START_UP
0020 OPEN (1)"MYCONFIG"
0030 READ (1,KEY=WHO,ERR=0050)X$
0040 SETFID X$
0050 CLOSE (1)
->?WHO
SMITHJ
```




5

Mnemonics

'@@'	'Cn'	'EM'	'JR'	'PICTURE'	'+T' & '-T'
'+\$' & '-\$'	'CAPTION'	'EO'	'JS'	'PIE'	'TEXT'
'2D'	'CE'	'EP'	'L6'	'PM'	'TEXTWDW'
'3D'	'CF'	'ER'	'L8'	'POLYGON'	'TR'
'4D'	'CH'	'ES'	'LC'	'POP'	'TW'
'AB'	'CI'	'ET'	'LD'	'PS'	'+U' & '-U'
'ARC'	'CIRCLE'	'EU'	'LF'	'PUSH'	'UC'
'AT'	'CL'	'EW'	'LI'	'*R'	'UP'
'+B' & '-B'	'COLOUR'	'+F' & '-F'	'LINE'	'RB'	'+V' & '-V'
'Bn'	'_COLOUR'	'Fn'	'LM'	'RC'	'VT'
'BACKGR'	'CP'	'FF'	'LPI'	'RECTANGLE'	'!W'
'BB'	'CPI'	'FILL'	'LT'	'RED'	'+W' & '-W'
'BE'	'CR'	'FL'	'MAGENTA'	'_RED'	'WA'
'BEEP'	'CS'	'FONT'	'_MAGENTA'	'RL'	'WC'
'BG'	'CURSOR'	'FRAME'	'MAXSIZE'	'RM'	'WD'
'BI'	'CYAN' '_CYAN'	'GD'	'MINSIZE'	'RP'	'WG'
'BJ'	'+D' & '-D'	'GE'	'ME'	'RS'	'WHITE'
'BK'	'DC'	'GF'	'MESSAGE'	'RT'	'_WHITE'
'BLACK'	'DEFAULT'	'GOTO'	'MINSIZE'	'+S' & '-S'	'WINDOW'
'_BLACK'	'DF'	'GREEN'	'MN'	'Sn'	'WM'
'BLUE'	'DIALOGUE'	'_GREEN'	'MODE'	'SB'	'WP'
'_BLUE'	'DN'	'GS'	'MOVE'	'SCROLL'	'WR'
'BM'	'DO'	'*H'	'MP'	'SE' & 'SD'	'WRAP'
'BO'	'DROP'	'HIDE'	'MS'	'SF'	'WS'
'BOX'	'+E' & '-E'	'*I'	'+N' & '-N'	'SHOW' / 'HIDE'	'WX'
'BR'	'EB'	'+I' & '-I'	'NI'	'SIZE'	'*X'
'BS'	'EE'	'IC'	'*O'	'SL'	'+X' & '-X'
'BT'	'EF'	'IMAGE'	'OFFSET'	'SN'	'XP'
'BU'	'EG'	'JC'	'OPTION'	'SP'	'YELLOW'
'BW'	'EI'	'JD'	'+P' & '-P'	'SR'	'_YELLOW'
'BX'	'EJ'	'JL'	'PE'	'SWAP'	'+Z' & '-Z'
'*C'	'EL'	'JN'	'PEN'	'SX'	'ZX'

Overview

ProvideX mnemonics deliver special control sequences to a display device or printer. Discussions on the creation, format, and use of mnemonics begin on the following page. For groupings see [Mnemonic Categories, p.581](#). For detailed descriptions, refer to the alphabetically-arranged [List of Mnemonics](#) on [p. 585](#).

Using Mnemonics

Mnemonics are generally inserted within a **PRINT** (or **INPUT**) statement to invoke such functions as clearing the screen, positioning the cursor, changing the colour of characters, setting/resetting various attributes, or enabling/disabling I/O modes.

All mnemonics are enclosed within single quotes. Some require arguments (e.g., `PRINT 'CIRCLE' (720,600,100,1)`). Some are represented by more than one keyword: a long form or short form (e.g., use either the **'PUSH'** or **'WC'** to copy the current window).

Use of an invalid mnemonic, or one that is not applicable to a particular device, results in `Error #29: Invalid Mnemonic or position specification`.



Note: Mnemonics are specific to the channel on which they are defined and are only valid while the channel remains open. When the channel is closed, the mnemonics are cleared.

Creating and Redefining Mnemonics

Use the **MNEMONIC** directive to define/redefine 2-character mnemonics for any file or device. For example, to assign settings for the ProvideX mnemonics **'CP'** and **'SP'**:

```
MNEMONIC(0)'CP'="Courier New,-8":120,40
MNEMONIC(0)'SP'="*":80,25
```

When a defined mnemonic is encountered in a **PRINT** or **INPUT** statement, the system converts it to the character string specified. Some 2-character mnemonics are predefined. For example, **'CR'**, **'LF'**, and **'FF'** are predefined as `00`, `$0A$`, and `$0C$` respectively.

All 2-character mnemonics listed in this reference that are used for character display can also be used for character printing. This is provided that the mnemonic is defined for the printer output channel using the correct escape sequence for the device specified.

Many motion/editing operations will be emulated when they are not actually supported by the device specified. However, it is still better to define mnemonics using the correct motion/editing sequences that are available for the specific device. Emulation of some sequences may result in slower performance.

For further details on mnemonic definitions, refer to the [MNEMONIC Directive, p.210](#), and the [\[WDX\] Tag, p.801](#).

X,Y Coordinates

When using GUI mnemonics, the values for x , y , and radius are considered logical screen coordinates not line/column coordinates. In ProvideX, these are known as graphical units (similar to pixels). Use `@X()` / `@Y()` functions, to convert line/column values to graphical units ; e.g.,

```
0110 LET RADIUS=(100)
0120 LET X=@X(45)! X=graphical units for column value 45
0130 LET Y=@Y(20)! Y=graphical units for line value 20
0140 PRINT @(45,15), 'CIRCLE'(X,Y,RADIUS,1)
```

Mnemonic Settings, Window / Region

When you send output to the screen, ProvideX remembers all graphic mnemonics until a '**CS**' [Clear Screen, p.598](#), is performed or the window is destroyed. This enables Windows to repaint the window properly when required.

Windows API Frame Styles

ProvideX has three Windows API frame styles available to it. The mnemonic '**WINDOW**' or '**WA**' [Mnemonics, p.647](#) uses **WS_BORDER** frame style, which is not popular with everyone. However, it has advantages: it maintains a global menu and buttons and is always relative to the main window. The mnemonic '**DIALOGUE**' [Mnemonic, p.600](#), on the other hand, uses **WS_DLGFAME**, which is more popular, but requires absolute screen position and doesn't allow global menus and buttons.

To change the look of the window frame without losing '**WINDOW**' functionality, you can use the **WS_THICKFRAME** style. To do this, add **OPT="Z"** to the '**WINDOW**' mnemonic:

```
PRINT 'WINDOW'(10,10,40,10,"Title",OPT="Z")
```

Dynamic Information in Mnemonics

Some mnemonics require dynamic information to generate specific output sequences (e.g., the line and column number in the '@@' mnemonic). Use the format below to define a mnemonic that contains dynamic information in its output sequence. You must identify a variable using a leading \ (backslash) followed by the one-character identifier, optional modifier, and format code.

Format *Define Variable:* `=ESC+"[\vd_char{modifier}fmt_code"`

Where

\vd_char Identifier for the variable. Include the backslash in the syntax. Valid identifiers include:

- \b Bottom margin of window/scroll region
- \h Height of window/scroll region
- \l (Lower case L). Left margin of window/scroll region
- \r Right margin of window/scroll region
- \t Top margin of window/scroll region
- \w Width of window/scroll region
- \A Current attributes in binary
- \B Background colour index
- \C Current column
- \F Foreground colour index
- \L Current line

modifier Optional modifier(s) of the variable's value. If you include these operators, insert them between the variable and the output *fmt_code*. You can include one or more of the modifiers. The list below shows the valid modifiers and their associated functions:

- + (Plus sign) Add the value of the next byte
- (Minus sign) Subtract the value of the byte
- & (Ampersand) AND the value of the next byte
- ^ (Caret) XOR the value of the next byte
- | (Pipe) OR the value of the next byte

fmt_code Mandatory output format code following each variable. Valid format codes include:

- 2 Output is two-byte ASCII
- 3 Output is three-byte ASCII
- a Output is ASCII plain text number (base 0)
- b Output is single byte binary (base 0)
- A Output is ASCII plain text number (base 1)
- B Output is single-byte binary (base 0x20)
- T Table output. Following T, the next byte must contain the number of bytes in the table. The table must follow that in the output sequence. If the number of bytes exceeds table length, no output is generated.

Mnemonic Categories

Mnemonics can be classified according to how they are used, and on the type of device they control. While there are exceptions where mnemonics can be redefined for use outside of their intended purpose, the following categories were created to help you identify the standard mnemonics by their *functionality* as well as their names:

Definition	Behaviour	Editing
GUI Display	Graphical Printer	Graphical Display/Printer
Character Display	Character Printer	Character Display/Printer
Motion		

Definition

Each of the following mnemonics contain a definition or provide information to be used by a device for performing specific operations:

'*C' Automatic Output on CLOSE	'*X' Program to Call on CLOSE
'*H' Control Screen Colours	'@@@' Define Cursor Position Sequence
'*I' Input Conversion Table	'AT' Character Attribute Output Sequence
'*O' Output Conversion Table	'GD' Define Graphics Character Set
'*R" OS Command String	'WX' Windows Definition Sequence

Behaviour

The following mnemonics are used to modify the behaviour of ProvideX specific to the channel on which they are defined:

'+E' & '-E' Multi-line Enter as Tab	'EL' Start Edit Key Load
'+F' & '-F' Signal Change of Focus	'EM' End Output Markup Mode
'+I' & '-I' Implied Decimals	'EO' End Output Transparency
'+N' & '-N' Drop Box Write Error	'ET' End Type Ahead
'+T' & '-T' Text Display On/Off	'EW' End WrapAround
'+U' & '-U' Screen Refresh	'FL' Start Function Key Load
'+V' & '-V' Row Highlighting	'IC' Insert a Space at Cursor
'+W' & '-W' Windows-Style	'LC' Mixed-Case User Input
'BEEP' Simple Sound Effect	'ME' Begin Edit Mode
'BG' Begin Generating Error #29	'MN' End Edit Mode
'BI' Begin Input Transparency	'OPTION' On-The-Fly Setting
'BM' Begin Markup	'RM' Reset to Default Mode
'BO' Begin Output Transparency	'SN' Native Screen Mode
'BT' Begin Type-Ahead Mode	'SX' Set Extended Screen Mode
'BW' Begin WrapAround	'WC' Save/Copy Current Window
'EG' End Generating Error #29	'WRAP' WrapAround On/Off
'EI' End Input Transparency	'ZX' Return Attributes as per BBx

Editing

The following mnemonics are used to control various editing operations in *both* character-based and GUI display environments:

'BE' Begin Echoing	'LI' Insert Line
'BI' Begin Input Transparency	'ME' Begin Edit Mode
'BT' Begin Type-Ahead Mode	'MN' End Edit Mode
'BW' Begin WrapAround	'NI' Next Input Numeric
'CE' Clear from Cursor to End of Screen	'RB' Ring Bell
'CF' Clear Foreground Mode	'RC' Return Cursor Address
'CI' Clear Input Type-Ahead Buffer	'RL' Return Line Contents
'CL' Clear from Cursor to End of Line	'RP' Terminal Read to End
'CS' Clear Screen	'RS' Restore Screen
'DC' Delete Character at Cursor	'SCROLL' Define/Control Scroll Region
'EE' End Echo Mode	'SE' & 'SD' Scroll Enable/Disable
'EI' End Input Transparency	'TR' Terminal Read from Start
'EL' Start Edit Key Load	'TW' Transmit Windows as String
'ET' End Type Ahead	'UC' Convert Input UpperCase
'EW' End WrapAround	'WRAP' WrapAround On/Off
'FL' Start Function Key Load	

Graphical Display/Printer

The following mnemonics are used all types of graphical output — they can be sent to a graphical print spooling system (i.e., *WINPRT*), or as output to a screen:

'ARC' Define/Draw Arc	'JD' Justify Decimal-Aligned
'Bn' Background Colour	'JL' Left-Justify Text
'BACKGR' or 'BK' Next Colour Is Background	'JN' Right-Justify Numeric
'BLACK' & '_BLACK' Colour Text	'JR' Right-Justify Text
'BLUE' & '_BLUE' Colour Text	'JS' Left-Justify String
'BU' Begin Underscoring	'LINE' Define / Draw a Line
'CIRCLE' Define / Draw a Circle	'LM' Landscape Mode
'CP' Condense Print for Screen	'LF' Line Feed (Advance Line)
'COLOUR' & '_COLOUR' User-Defined Colours	'MAGENTA' & '_MAGENTA' Colour Text
'CYAN' & '_CYAN' Colour Text	'MS' Mode Serial
'DEFAULT' or 'DF' Define Default	'PEN' Define Pen Style
'EP' Start Expanded Print	'PICTURE' Define / Draw Picture
'EU' End Underscoring	'PIE' Define / Draw Pie Slice
'FILL' Define Fill Style	'PM' Portrait Mode
'Fn' Foreground Colour	'POLYGON' Draw Polygon
'FONT' Define / List Fonts	'RECTANGLE' Draw a Rectangle
'GREEN' & '_GREEN' Colour Text	'RED' & '_RED' Colour Text
'GE' End Graphics Data	'SP' Standard Print
'GS' Start Graphics Transmission	'TEXT' Draw Text
'IMAGE' Define a Graphics Group	'WHITE' & '_WHITE' Color Text
'JC' Justify Centre	'YELLOW' & '_YELLOW' Colour Text

For more graphical mnemonics, see [GUI Display](#) and [Graphical Printer, p.583](#).



Note: In Windows, the 'FILL', 'FONT', 'PEN', and 'PICTURE' mnemonics use Graphical Device Interface (GDI) resources/handles that are only released when the window they are in is dropped or cleared, or their 'IMAGE' group is deleted.

Graphical Printer

The following mnemonics are used only when sent to a graphical printer spooling system (i.e., *WINPRT*):

'+S' & '-S' Substitute Solid Lines On/Off	'L8' Set to 8 LPI
'AB' Abort (For Windows Spooler)	'LPI' Logical Lines / Inch
'CPI' Logical Characters per Inch	'OFFSET' Offset for *WINPRT*
'L6' Set to 6 LPI	

For more graphical mnemonics, see [Graphical Display/Printer, p.582](#).

GUI Display

The following mnemonics are used only for graphical user interfaces:

'+E' & '-E' Multi-line Enter as Tab	'GF' Default Window Font
'+F' & '-F' Signal Change of Focus	'GOTO' Make Window Current
'+I' & '-I' Implied Decimals	'GREEN' Colour Text
'+N' & '-N' Drop Box Write Error	'HIDE' Hide Window Display
'+P' & '-P' Mouse Movements	'MAXSIZE' Window Resize Limit
'+T' & '-T' Text Display On/Off	'MESSAGE' Message Line Text
'+U' & '-U' Screen Refresh	'MINSIZE' Window Resize Limit
'+V' & '-V' Row Highlighting	'MOVE' Move Current Window
'+W' & '-W' Windows-Style	'POP' Restore Previous Window
'+X' & '-X' Windows 'X' Close Button	'PUSH' Save/Copy Window
'+Z' & '-Z' Text-to-Windows Look	'SCROLL' Define Scroll Region
'2D' Use 2D Controls	'SE' & 'SD' Scroll Enable/Disable
'3D' Use 3D Controls	'SHOW' Show Window Display
'4D' Use 4D Controls	'SIZE' Visual Size of Window
'BOX' Define / Draw a Box	'SR' Scroll Reset
'BR' Begin Reverse Video	'SWAP' Swap Windows on Stack
'BX' Define / Draw a Box	'TEXTWDW' Text Window
'CAPTION' Replace Caption	'WA' Define / Draw Window
'Cr' Control Cursor Display	'WC' Save/Copy Current Window
'CURSOR' Cursor/Mouse Pointer	'WG' Make Window Current
'DIALOGUE' Define Dialogue	'WINDOW' Define / Draw Window
'DO' Delete Objects in Scroll Region	'WR' Remove Current Window
'DROP' Drop Identified Window	'WS' Swap Windows On Stack
'ER' End Reverse Video	
'FRAME' Define / Draw a Frame	

For more graphical mnemonics, see [Graphical Display/Printer, p.582](#).

Character Printer

The following mnemonics are used only when sent to direct-to-output devices:

'EF' End Expanded Print	'PS' Enable Attached Printer Port
'EL' Start Edit Key Load	'Sn' Slew to Channel #
'EP' Start Expanded Print	'SL' Start VFU Load
'L6' Set to 6 LPI	'VT' Slew to S6, Vertical Tab
'L8' Set to 8 LPI	'WP' Wide Printer (DOS)
'PE' Disable Attached Printer Port	

Character Display

The following mnemonics are used only when sent to text-based CUI devices:

'Bn' Background Colour	'MODE' Set Attributes/Colour
'BACKGR' or 'BK' Next Colour Is Background	'MOVE' Move Current Window
'BB' Begin Blinking (DOS)	'POP' Restore Previous Window
'BJ' Join Box Intersections	'PUSH' Save/Copy Window
'BLACK' & '_BLACK' Colour Text	'RED' & '_RED' Colour Text
'BLUE' & '_BLUE' Colour Text	'RS' Restore Screen
'BOX' Define / Draw a Box	'SB' Set Mode to Background
'BX' Define / Draw a Box	'SCROLL' Define Scroll Region
'CAPTION' Replace Caption	'SE' & 'SD' Scroll Enable/Disable
'Cn' Control Cursor Display	'SF' Set Mode to Foreground
'COLOUR' & '_COLOUR' User-Defined Colours	'SR' Scroll Reset
'CURSOR' Cursor/Mouse Pointer	'SWAP' Swap Windows on Stack
'CYAN' & '_CYAN' Colour Text	'TEXTWDW' Text Window
'DEFAULT' or 'DF' Define Default	'WA' Define / Draw Window
'DROP' Drop Identified Window	'WC' Save/Copy Current Window
'EB' End Blinking Mode (DOS)	'WD' Drop Identified Window
'EJ' End Box Joining	'WG' Make Window Current
'EP' Start Expanded Print	'WHITE' & '_WHITE' Color Text
'Fn' Foreground Colour	'WINDOW' Define / Draw Window
'GE' End Graphics Data	'WM' Relocate Current Window
'GOTO' Make Window Current	'WR' Remove Current Window
'GS' Start Graphics Transmission	'WS' Swap Windows On Stack
'GREEN' & '_GREEN' Colour Text	'XP' Line Mode (DOS)
'MAGENTA' & '_MAGENTA' Colour Text	'YELLOW' & '_YELLOW' Colour Text

Character Display/Printer

The following mnemonics are usable on *both* direct-to-output and text-based CUI devices:

'BR' Begin Reverse Video	'ES' Send Escape
'BU' Begin Underscoring	'FF' FormFeed
'CP' Condense Print for Screen	'LF' Line Feed (Advance Line)
'ER' End Reverse Video	'MS' Mode Serial
'EU' End Underscoring	'SP' Standard Print

Motion

The following mnemonics are used to direct cursor movement in *both* text-based and GUI display environments:

'BS' Cursor Back One Space	'DN' Move Cursor Down a Line
'CH' Position Cursor at Home	'LT' Move Left One Column
'CR' Carriage Return	'RT' Move Right One Column
'CS' Clear Screen	'UP' Move Up One Line

List of Mnemonics

The ProvideX mnemonics listed below are in ASCII sort order, except where the mnemonic keyword has an initial asterisk, plus sign, minus sign or underscore. These are listed under their second character; e.g., you'll find '**_BLUE**' under "**B**" with '**BLUE**' and '**+S**' / '**-S**' under "**S**".

'@@' Mnemonic

Define Cursor Position Sequence

Definition

Format '@@'=esc_seq\$

Where:

esc_seq\$ ESC+"[" and a string expression defining the cursor position coordinates, using the format:

`\vd_char{modifier | ;}format_code`

For example, '@@'=ESC+" [\LA;\CAH".

Description Use '@@' to define the cursor print positioning.

For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

'+\$' & '-\$' Mnemonics

For Internal Use Only

GUI Display

Description For Sage Software Canada use only - Internal debugging attribute included here for completeness only.

'2D' Mnemonic

Use 2D Controls

GUI Display

Description Display two-dimensional controls (older Windows visual style) for all current/subsequent windows (deactivates '3D' or '4D' controls).

'3D' Mnemonic*Use 3D Controls***GUI Display**

Description Display three-dimensional controls (Windows 95 visual style) for all current/subsequent windows. To deactivate, use **PRINT '2D'**.

'4D' Mnemonic*Use 4D Controls***GUI Display**

Description Display Windows Vista (or XP) visual style for all current/subsequent windows. Vista style appears only when run under Vista. On other Windows systems, this mnemonic renders an XP visual style. To deactivate, use **PRINT '2D'**.

'AB' Mnemonic*Abort (For Windows Spooler)***Graphical Printer**

Description For Windows Spooler Only. Use **'AB'** to abort output to the Windows print spooler. See also **Printing in Windows**, *User's Guide*.

Example `PRINT (30) 'ab'`

'ARC' Mnemonic*Define/Draw Arc***Graphical Display/Printer**

Format `'ARC'(x,y,radius,aspect,angle_1,angle_2)`

Where:

angle_1 Starting angle, in degrees. Numeric expression.

angle_2 Ending angle, in degrees. Numeric expression.

aspect Aspect ratio / viewpoint. (Ratio=1 results in no tilt.) Numeric expression.

radius Radius of the circle, in graphical units. Numeric expression.

x,y Coordinates for the centre of the drawing, in graphical units. Numeric expression.

Description Use 'ARC' to draw (print) an arc on the device. Use graphical units or **@X(col)** and **@Y(line)** functions for x , y , and *radius*. The arc will extend from the starting *angle1* to *angle2*. The 'ARC' mnemonic uses the current 'PEN' attributes.

Example The example below draws two different-coloured arcs. Aspect ratio=3 tilts the resulting drawing into an elliptical "shape". (Since this ellipsis is really two arcs as opposed to an enclosed shape, 'FILL' attributes do not apply.) Aspect ratio=1 would join the arcs in a circular shape instead.

```
0060 PRINT 'PEN' (1,3,1)
0070 PRINT 'ARC' (800,250,75,3,1,75)
0080 PRINT 'PEN' (
0090 PRINT 'ARC' (800,250,75,3,75,1)
```

'AT' Mnemonic Character Attribute Output Sequence

Definition

Description Use 'AT' to define a character attribute output sequence.

For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

'+B' & '-B' Mnemonics Output Buffering On/Off

Behaviour or GUI Display

Format *Enable File Buffering (default): '+B'*
Disable File Buffering: '-B'

Description Setting this option enables file output to be buffered or not buffered (physically flushed) after each **PRINT** or **WRITE** operation. The primary use of this option is for Log type files.

'Bn' Mnemonic*Background Colour***Graphical Display/Printer or Character Display***Where:*

n Numeric colour code. Default is **'B0'**. Supported options are:

0 Black*	4 Blue	8 Dark Gray	< Light Blue
1 Red	5 Magenta	9 Light Red	= Light Magenta
2 Green	6 Cyan	: Light Green	> Light Cyan
3 Yellow	7 White	; Light Yellow	? Light Gray

Description Use the format above to begin outputting a background colour to a device (i.e., **PRINT**, **INPUT**, etc). The 16 colour codes are in ASCII sequence from 0 to ?.

Use the same colour codes for **'Fn'** **Foreground Colour**, *p.607*.

Example In this example, the prompt and user's response will both be in white text on a blue background.

```
0010 INPUT 'B4', 'WHITE', "Please enter your name: ", NAME$,
```

'BACKGR' or **'BK'** Mnemonic *Next Colour Is Background***Graphical Display/Printer or Character Display**

Format *Long or short form: 'BACKGR' or 'BK'*

Description Use either **'BACKGR'** or **'BK'** to designate that the next colour mnemonic (e.g., **'BLUE'**) is the background colour for the text region when a directive sends output to the display device (i.e., **PRINT**, **INPUT**, etc.).

Example

```
0020 PRINT 'BK', 'BLUE', 'WHITE', "Hello, ", USER_ID$,
```

'BB' Mnemonic*Begin Blinking***Character Display**

Description *Included for completeness only.* Use **'BB'** to begin blinking of characters (legacy DOS systems). To end blinking mode, use the **'EB'** Mnemonic, *p.603*.

'BE' Mnemonic

Begin Echoing

Editing

Description Use 'BE' to begin echoing characters entered from the keyboard in response to an **INPUT** (or **READ**, **FIND**, ...) directive. Echoing is **ON** by default for *channel 0* (zero, the main terminal). For other channels, echoing is **OFF** by default. To end echoing, use the 'EE' Mnemonic, *p.603*.

'BEEP' Mnemonic

Simple Sound Effect

Behaviour

Description The 'BEEP' mnemonic allows applications to play sounds in a simpler manner than the **MULTI_MEDIA** command. This mnemonic accepts a string that contains the same keywords as the **MSGBOX** directive relative to the system sounds of "STOP", "QUESTION", "INFO", "EXCLAMATION", "?" and "!"; e.g.,

```
PRINT 'BEEP' ("STOP")
PRINT 'BEEP' ("*STOP,w")
```

The *,w* option causes ProvideX to *wait* until the sound has completed before continuing any processing. In order for *,w* to work, the sound needs to be registered. Use an *asterisk* * before the sound name to have the system look up the sound in the Windows registry (which is not the same as sound names in the control panel); e.g.,

```
PRINT 'BEEP' ("*MailBeep")      ! Registered Sound Event
PRINT 'BEEP' ("*C:\Windows\Media\tada.wav") ! Specific File
PRINT 'BEEP' ("*Windows XP Logoff Sound") ! Found by automatic search
```

In the last example, the real filename is C:\Windows\Media\Windows XP Logoff Sound.wav. If the name given does not match a registered sound, then the Windows OS searches its list of possible sound locations (current directory, Windows directory, System directory, etc.). For more details, consult the Microsoft documentation on *Registering Sound Events* or check the registry for `HKEY_CURRENT_USER\AppDataEvents\EventLabels*`

'BG' Mnemonic

Begin Generating Error #29

Behaviour

Description Use 'BG' to begin generating Error #29: Invalid Mnemonic or position specification whenever an invalid mnemonic is executed. To end generation of Error #29, use the 'EG' Mnemonic, *p.603*.

'BI' Mnemonic*Begin Input Transparency***Editing or Behaviour**

Description Use 'BI' to begin input transparency mode. (All input is accepted without any system interpretation of control codes. With 'BI' activated, a **SIZ=** option on an **INPUT** directive is the only way you can specify a terminator / end of input. To end input transparency mode, use the **'EI' Mnemonic, p.604**.

Example When you need to return input one character at a time, we recommend that you use the **'ME' Mnemonic, p.620**. However, it is also possible to use 'BI' in combination with **SIZ=1** to return input one character at a time. Then, each character is returned as entered. Standard characters are returned in the variable specified (e.g., CHAR\$, below). Function and edit keys are returned in the system variable **CTL**. Your program will still be responsible for processing all the edit keys.

```
0010 INPUT (0,SIZ=1) 'BI', CHAR$, 'EI' ! This will echo the data
0010 OBTAIN (0,SIZ=1) 'BI', CHAR$, 'EI' ! This will not echo
```

'BJ' Mnemonic*Join Box Intersections***Character Display**

Description Use 'BJ' to tell ProvideX to begin joining box intersections. After you use this mnemonic, all 'BOX' edge lines which intersect existing graphic line characters will be adjusted to join the intersecting lines properly. To end box joining, use the **'EJ' Mnemonic, p.604**.

Example

```
0030 PRINT 'BOX'(4,6,16,10,"Box 1",'BJ'),
0040 PRINT 'BOX'(19,6,10,10,"Box 2"),
0050 PRINT 'EJ',
```

'BK' Mnemonic*Next Colour Is Background***Graphical Display/Printer or Character Display**

Description See **'BACKGR' Next Colour Is Background, p.588**.

'BLACK' & '_BLACK' Mnemonics*Colour Text***Graphical Display/Printer or Character Display**

Format *Foreground:* 'BLACK'
 Background: '_BLACK'

Description All input or output following this mnemonic will be in black foreground or background.

Example INPUT '_CYAN', 'BLACK', "Please enter your name: ", Name\$,

'BLUE' & '_BLUE' Mnemonic*Colour Text***Graphical Display/Printer or Character Display**

Format *Foreground:* 'BLUE'
 Background: '_BLUE'

Description All input or output following this mnemonic will be in blue foreground or background.

Example INPUT '_BLUE', 'WHITE', "Please enter your name: ", Name\$,

'BM' Mnemonic*Begin Output of Markup Files***Behaviour**

Description Use 'BM' to begin output of markup files containing embedded mnemonics. For instance, in the ProvideX *VIEWER*, 'BM' sends all data directly to the print file without interpretation in tokenized format (except for 'FF', 'CR', and 'LF' mnemonics, which are output as 0C, 0D, and 0A respectively). This allows you to send print jobs to any Windows printer. To end markup, use the 'EM' Mnemonic, [p.605](#).

'BO' Mnemonic*Begin Output Transparency***Behaviour**

Description Use 'BO' to begin output transparency mode. (All printed output is sent directly to the display device without checking for embedded mnemonics; i.e., \$1B\$ (Esc) followed by the mnemonic code.) Use 'EO' to end output transparency mode.



'BOX' Mnemonic

Define / Draw a Box

GUI Display or Character Display

- Format** *Long or short form: 'BOX' or 'BX'*
'BOX'(col,ln,wth,ht[,title\$[,attrib\$]])
- Where:*
- @(col,ln,wth,ht)** Position/coordinates. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.
- attrib\$** Optional attributes. If you include attributes, use a string of one or more mnemonics.
- title\$** Optional title. String expression.
- Description** You can use either **'BOX'** or **'BX'** in the format to draw a text mode box. If you include a title, it is displayed left-justified on the top line of the box unless the **'AH'** system parameter is set. If you include attributes, these are sent to the display device before the box is displayed. Refer to the **'AH' System Parameter, p.656**.
- Example** The boxes in the example are drawn joined with titles. Current **'FILL'** and **'PEN'** settings are ignored when you use **'BOX'**:

```
0010 PRINT 'CS'; LIST
0020 PRINT 'PEN'(2,3,6), 'FILL'(3,8)
0030 PRINT 'BOX'(4,6,16,10,"Box 1", 'BJ')
0040 PRINT 'BX'(19,6,10,10,"Box 2")
0050 PRINT 'EJ'
```

'BR' Mnemonic

Begin Reverse Video

GUI Display or Character Display/Printer

- Description** Use **'BR'** to begin sending characters to the device in reverse video mode. Use an **'ER'** mnemonic to end reverse video mode. (When you have colour attributes, ProvideX uses the background colour as the foreground colour and vice-versa.) To end reverse video mode, use the **'ER' Mnemonic, p.605**.

'BS' Mnemonic

Cursor Back One Space

Motion

- Description** Use **'BS'** to move the cursor back one position to the left. ProvideX ignores this mnemonic if the cursor is in column 0 *zero*.

'BT' Mnemonic*Begin Type-Ahead Mode***Behaviour or Editing**

Description Use **'BT'** to begin type-ahead mode. This mode lets the user enter input before the execution of an **INPUT** statement. ProvideX supplies an internal buffer for the characters entered. To end type-ahead mode, use **'ET' End Type Ahead**, *p.606*. (See also: **'CI' Clear Input Type-Ahead Buffer**, *p.595*.)

'BU' Mnemonic*Begin Underscoring***Graphical Display/Printer or Character Display/Printer**

Description Use **'BU'** to begin underscoring characters sent to the display device. To end underscoring, use the **'EU'** Mnemonic, *p.606*.

'BW' Mnemonic*Begin WrapAround***Behaviour or Editing**

Description Begin WrapAround. (See **'EW'** or **'WRAP' WrapAround On/Off**, *p.648*.)

'BX' Mnemonic*Define / Draw a Box***GUI Display or Character Display**

Description See **'BOX' Define / Draw a Box**, *p.592*.

'*C' Mnemonic*Automatic Output on CLOSE***Definition**

Format **MNEMONIC** (*chan*)*C'=data\$

Description '*C' (*star-c*) contains a string expression that will be automatically printed to the device driver when a **CLOSE** is executed.



Note: For ***WINPRT*** devices, because the Windows spooler automatically resets the printer when closed, the '*C' mnemonic is ignored.

For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

'Cn' Mnemonic*Control Cursor Display Mode***GUI Display or Character Display**

Where:

<i>n</i>	Display mode:	
	0 - zero	Hide Cursor 'C0'
	1 - Display	Normal Cursor 'C1'
	2 - Insert	Mode Cursor 'C2'

Description Use the format above ('C0', 'C1', or 'C2') to control how the cursor is displayed.

'CAPTION' Mnemonic*Replace Caption for Window***GUI Display**

Format 'CAPTION'(text)

Where:

text Replacement caption. String expression.

Description Use 'CAPTION'(text) to change the caption for the current window. If the current window does not have a caption and is located at position 0.0, ProvideX replaces the caption on the main window. ('WINDOW' or 'DIALOGUE')

To find out the caption for the current window you can use a **MULTI_LINE READ** directive with CTL=0, as in `MULTI_LINE READ 0,A$`.

Example

```
0010 multi_line read 0,a$
0020 print a$
0030 print 'caption'("My Window")
0040 multi_line read 0,a$
0050 print a$
```

'CE' Mnemonic *Clear from Cursor to End of Screen*

Editing

Description Use 'CE' to clear the screen from the cursor position to the end of the screen.

'CF' Mnemonic *Clear Foreground Mode*

Editing

Description Use 'CF' to clear non-protected mode for characters on the screen. (See also: '[SF' Set Mode to Foreground, p.639.](#))

'CH' Mnemonic *Position Cursor at Home*

Motion

Description Use 'CH' to position the cursor to the device's home location; i.e., @(0 , 0). The screen is not cleared.

'CI' Mnemonic *Clear Input Type-Ahead Buffer*

Editing

Description Use 'CI' to clear any input from the type-ahead buffer. (See also '[BT' Begin Type-Ahead Mode, p.593.](#))

'CIRCLE' Mnemonic*Define / Draw a Circle***Graphical Display/Printer**

Format **'CIRCLE'**(*x,y,radius,aspect*)

Where:

aspect Numeric aspect ratio / viewpoint. (Ratio=1 results in no tilt.)

radius Radius of the circle, in graphical units. Numeric expression.

x,y Coordinates for the centre of the drawing, in graphical units.
Numeric expression.

Description Use **'CIRCLE'** to draw (print) a circle on the device. For *x, y*, and *radius*, use graphical units or the **@X()** / **@Y()** Functions, [p.391](#). The **'CIRCLE'** mnemonic uses the current attributes for **'FILL' Define Fill Style**, [p.607](#), and **'PEN' Define Pen Style**, [p.630](#).

Example

```
0170 PRINT 'PEN' (1,3,1), 'FILL' (2,6)
0180 PRINT 'CIRCLE' (720,600,100,1)
0190 PRINT 'PEN' (1,3,6), 'CIRCLE' (950,600,100,2)
```

'CL' Mnemonic*Clear from Cursor to End of Line***Editing**

Description Use **'CL'** to clear the current line from the cursor position to the end of the line.

'COLOUR' & '_COLOUR' Mnemonics *User-Defined Colours***Graphical Display/Printer or Character Display**

Format *Foreground:* **'COLOUR'** ("RGB: *n n n*" | *num* | *name\$*)

'COLOR' ("RGB: *n n n*" | *num* | *name\$*)

Background: **'_COLOUR'** ("RGB: *n n n*" | *num* | *name\$*)

'_COLOR' ("RGB: *n n n*" | *num* | *name\$*)

Where:

name\$ Pre-defined or user-defined colour name; e.g., 'COLOUR' ("LightBlue").

num Pre-defined (0-15) or user-defined (16-254) colour number (*n*=0-254).

RGB: *n n n* Three-number RGB value (*n*=0-255).

Description All input or output following this mnemonic will be set to the defined foreground or background colour. For further information, refer to the description for changing the colour index provided under the **'OPTION' Mnemonic**, [p.627](#).

'CP' Mnemonic*Condense Print for Screen***Graphical Display/Printer or Character Display/Printer**

Description Use **'CP'** to reduce the window and region size, and/or change the font to condensed print. To restore the screen and font to regular size, use the **'SP'** Mnemonic, *p.640*. In ProvideX, **'CP'** or **'SP'** affect only the data that follows the mnemonic.

On GUI devices, the **'CP'** mnemonic will output using a font size that is approximately 5/8ths the size of the current default text font.

Example

```
10 open(1) "LP"
20 print (1) "NORMAL"+ ' CP ' + "COMPRESSED" + ' SP ' + "NORMAL"
30 close(1)
```

If the example above is run in ProvideX, the word COMPRESSED would be in condensed print, and NORMAL would be in standard print. (In other Business Basics, the font for the complete line is affected and, for this example, would be in standard print.)

'CPI' Mnemonic*Logical Characters per Inch***Graphical Printer**

Format **'CPI'**(chars)

Where:

chars Logical characters per inch. Numeric expression.

Description Use **'CPI'**(chars) to set the logical CPI (characters per inch) for printing in graphics mode, where (as a rough guide to equivalent sizes):

Point size -12 = 6 LPI, 10 CPI

Point size -10 = 7.2 LPI, 12 CPI

Point size -7 = 10 LPI, 16 CPI

For more information refer to **'LPI'** **Logical Lines / Inch**, *p.618*, and the **TXH()** **Function**, *p.544*, and the **TXW()** **Function**, *p.545*.

Example

With both channels (30) and (1) open:

```
0060 PRINT (30) 'CPI' (120/7.5), 'LPI' (6), Data$
0070 PRINT (1) 'CPI' (16), Data$20 print
```

'CR' Mnemonic*Carriage Return***Motion**

Description Use 'CR' to return to column 0 (i.e., Carriage Return without line feed).

'CS' Mnemonic*Clear Screen***Editing or Motion**

Description Use 'CS' to clear the screen and set the cursor position @ (0 , 0).

As alternatives, you can use the 'CH' (Cursor Home without clear screen) and 'FF' (FormFeed, alternative to clear screen for print data) mnemonics.

'CURSOR' Mnemonic*Control Cursor, Mouse Pointer***GUI Display or Character Display****Formats**

1. *Cursor ON (Default):* 'CURSOR' ("ON")
2. *Hide Cursor:* 'CURSOR' ("OFF")
3. *Cursor in Replace Mode:* 'CURSOR' ("REP")*
4. *Cursor in Insert Mode:* 'CURSOR' ("INS")
5. *Change Mouse Pointer (GUI Only):* 'CURSOR' (num)

Where:

num Numeric code for graphic to use as mouse pointer. (Graphics display only.)

Supported options include:

- | | |
|--|--|
| 0 - Arrow | 7 - Rabbit in hat |
| 1 - Wait (Hourglass) | 8 - Happy face |
| 2 - I-Beam | 9 - Sad face |
| 3 - Movement arrows | 10 - Resize vertical Up/Down arrow |
| 4 - Sizing arrow | 11 - Resize horizontal Left/Right arrow |
| 5 - Hand | 12 - Not allowed (diagonal line across circle) |
| 6 - Hand in crossed circle ("No" hand) | |

Description Use the above formats to control cursor and mouse pointer displays.



Note: The mouse pointer selected via the 'CURSOR' mnemonic may be overridden by settings in the INI file.

Example

```

0010 PRINT 'CS'
0020 CUR=1
0030 BUTTON 10,@(10,10,20,2)="CHANGE CURSOR"
0040 WHILE CTL<>4
0050 OBTAIN X
0060 IF CTL=10 THEN PRINT 'CURSOR'(CUR)
0070 IF CUR=12 THEN CUR=0 ELSE CUR++
0080 WEND
0090 PRINT 'CURSOR'(0)
0100 END

```

'CYAN' & '_CYAN' Mnemonics

Colour Text

Graphical Display/Printer or Character Display

Format *Foreground: 'CYAN'*
Background: '_CYAN'

Description All input or output following this mnemonic will be in cyan foreground or background.

Example INPUT '_CYAN','WHITE',"Please enter your name: ",Name\$,

'+D' & '-D' Mnemonics

Obsolete

Behaviour or GUI Display

Description *Included here for completeness only.*

'DC' Mnemonic

Delete Character at Cursor

Editing

Description Use 'DC' to delete the character at the current cursor position. (Text shifts one position to the left for all characters to the right of the cursor on the same line.)

'DEFAULT' or 'DF' Mnemonic*Define Default***Graphical Display/Printer or Character Display**

Format *Long or short form: 'DEFAULT' or 'DF'*

Description Use either **'DEFAULT'** or **'DF'** to define the current attributes/font as the default for an **OPEN** channel. For instance, you can set a default for fixed font, reverse video, blinking, underscore, foreground / background, etc. To set the default font for all graphical objects, refer to the **'GF' Mnemonic, p.612**.

Example To set a font in ***WINPRT*** printing as the standard default font for an **OPEN** channel:

```
0010 OPEN (30) "*WINPRT*;ASIS"
0020 PRINT (30) 'FONT' ("Courier New",-7), 'DF'
```

'DIALOGUE' Mnemonic*Define/Draw Dialogue Region***GUI Display**

Format **'DIALOGUE'** (*col,ln,wth,ht[,wdw_id],[title\$][,attrib\$][,OPT=string\$]*)

Where:

col,ln,wth,ht Position / coordinates. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.

attrib\$ Optional attribute string. If you include attributes, use one or more mnemonics to define the defaults for the window. String expression.

title\$ Optional title. String expression.

string\$ Optional attributes. Supported options include:

- &** Ampersand - creates window that logically attaches to the current window (i.e., leaves the current window active and shares controls)
- *** Asterisk - creates resizable window with automatic scrollbars for text plane; e.g., `PRINT 'DIALOGUE' (1,1,60,20, "Title", OPT="*")`
- Dash - window has a minimize button.
- ?** Window supports Win95 *Help* button.
- ^** Caret - window is always on top (not applicable to the **'WINDOW'** mnemonic); e.g., `PRINT 'DIALOGUE' (1,2,30,3, "My Top Dog", OPT="^")`.
- c** New window is a child of the window that launched it.
- C** Disables close button on title bar of window and eliminates the system control menu from the title bar.
- F** Window can be maximized, occupying full screen (regardless of number of columns/rows). Area outside the defined text region will be cleared to the default background colour for the window.

- h** Window has no title bar.
- i** Window has no icon in the upper left corner.
- m** Enables "maximize" box in top right corner of window (only for dialogue windows created with `OPT=" * * "`).
- M** Window has a menu bar.
- s** **'DIALOGUE' ()** returns a CTL value to signal when the dialogue view state is changed; i.e.,
 CTL=-1106 when the dialogue is minimized
 CTL=-1107 when the dialogue is restored to normal state
 CTL=-1105 when the dialogue is maximized.
 (CTL=-1105 is also returned for the **Z** option (below) if dialogue has been resized.)
- S** Window has *Status line / Message Bar*.
- x** Disables close button on title bar of window and eliminates the system control menu from the title bar.
- X** Enables close button on title bar of window and supports the system control menu on the title bar.
- Z** Creates a resizable window *without scrollbars*. When the user resizes the window a CTL=-1105 is generated. See also **'SIZE'**, p.639.

`wdw_id` Optional dialogue window's unique ID number (0 -255).

Description

Use **'DIALOGUE'** to define a new window which is not contained in the main ProvideX screen. In a non-Windows environment, this does the same as a **'WINDOW'** mnemonic.



Note: The **'DIALOGUE'** mnemonic with `OPT=" ^ "` can be useful for error message windows. Although the users can still perform other operations, the error message will remain "always on top" as a constant reminder to deal with the error.

ProvideX uses the `WS_DLGFRAME` frame style from the Windows API for the **'DIALOGUE'** mnemonic. For more information, see **Windows API Frame Styles**, p.579.

See also: **'TEXTWDW' Create Text Window**, p.643 and **'WINDOW' Define / Draw Window**, p.647.

Example

The example below creates a dynamic, resizable/scrollable viewer using a **'DIALOGUE'** window which is large enough to display the complete picture. A **'SIZE'** mnemonic fits the window to the screen. The user is supplied with scroll bars to view the desired image.

```
0010 CALL  "*picture;Get_size", "c:\windows\clouds.bmp", WD, HI
0020 LET  WD=INT(WD), HI=INT(HI)
0030 PRINT 'DIALOGUE' (10,10,WD,HI, "MY Photo", OPT=" *-m" ),
0030:  'SIZE' (30,10), 'B?', 'SR', 'CS',
```

(ProvideX normally redraws the text plane below the picture. Line 0040 suppresses the text plane to avoid flicker.)

```
0040 PRINT '-T',
0050 PRINT 'PICTURE' (0,0,@X(WD),@Y(HI), "C:\windows\clouds.bmp" ),
0060 OBTAIN 'C0',*; IF CTL<>4 THEN GOTO *SAME
```

'DN' Mnemonic*Move Cursor Down a Line***Motion**

Description Use 'DN' to move the cursor down a line.

'DO' Mnemonic*Delete Objects in Scroll Region***GUI Display**

Description Use 'DO' to delete all objects that start (top left corner) in the current scroll region.

'DROP' or 'WD' Mnemonic*Drop Identified Window***GUI Display or Character Display**

Format *Long or short form: 'DROP' or 'WD'*
'DROP'(wdw_id)

Where:

wdw_id Window's unique ID number (0 - 255).

Description Use either 'DROP' or 'WD' in the format to drop the identified window. If this window does not exist or is the only window, ProvideX returns an Error #57: No such window defined. If *wdw_id*=-1, all windows except the primary window are dropped.

Example

```
30400 LET WW_ADD=HWN(0)
30410 PRINT %W_MSG$, 'WINDOW' (20,8,40,8,WW_ADD,"Deleting Sku"
30410:), 'CS', 'SB',
30420 LET DEL_SKU$=FROM_SKU$, SAVE_KEY$=COMP$+FORM$
30430 !
30440 LET ADD_SKU$=TO_SKU$
30460 GOSUB DELETE_ITEM
30470 READ (E855FENT_H,KEY=SAVE_KEY$)IOL=0310
30480 PRINT 'WG' (W_ADD), 'WD' (WW_ADD), %NORM_SCR$,
```

'+E' & '-E' Mnemonics*Multi-line Enter as Tab***Behaviour or GUI Display**

Format *Allow* `Enter` = `Tab` : '+E'

Disable `Enter` = `Tab` : '-E'

Description Use '+E' and '-E' to control the use of the `Enter` key as `Tab` in multi-lines.

'EB' Mnemonic*End Blinking Mode (DOS)***Character Display**

Description *Included for completeness only.* Use 'EB' to end blinking of characters (legacy DOS systems). This is the opposite of the **'BB' Mnemonic, p.588**.

'EE' Mnemonic*End Echo Mode***Editing**

Description Use 'EE' to end echoing of characters sent to the display device. This is the opposite of the **'BE' Mnemonic, p.589**.

'EF' Mnemonic*End Expanded Print***Graphical Display/Printer or Character Printer**

Description Use 'EF' to end expanded print mode. This is the opposite of the **'EP' Mnemonic, p.605**.

'EG' Mnemonic*End Generating Error #29***Behaviour**

Description Use 'EG' to stop generation of Error #29: Invalid Mnemonic or position specification whenever an invalid mnemonic is executed. This is the opposite of the **'BG' Mnemonic, p.589**.

'EI' Mnemonic*End Input Transparency***Editing or Behaviour**

Description *Default.* Use 'EI' to end input transparency mode. This is the opposite of the 'BI' Mnemonic, [p.590](#).

'EJ' Mnemonic*End Box Joining***Character Display**

Description *Default.* Use 'EJ' to end automatic joining of box intersections for 'BOX' mnemonic drawings. This is the opposite of the 'BJ' Mnemonic, [p.590](#).

Example

```
0030 PRINT 'BOX' (4,6,16,10,"Box 1",'BJ')
0040 PRINT 'BOX' (19,6,10,10,"Box 2")
0050 PRINT 'EJ'
```

'EL' Mnemonic*Start Edit Key Load***Editing or Behaviour**

Description Use 'EL' to start loading edit keys. This feature is included for compatibility with other languages. For more information on conversion and compatibility modes, see the *Installation Guide*. Also, refer to the [DEFCTL Directive, p.79](#).

ProvideX Utilities:

ProvideX utilities do not expect function and editing keys to be loaded with other values from the use of 'FL' or 'EL'. Issue a `PRINT 'FL', "1", 'EL', "1"` to reset loaded function or editing keys just prior to running any of the ProvideX utilities. See also: '[FL' Start Function Key Load, p.608](#).


'EL' Mnemonic*End VFU Load***Character Printer**



Description Use 'EL' to end VFU load. The data following 'SL' (from 'SL' up to an 'EL') defines the VFU channels. The total number of characters defines the page length, the characters themselves represent the channels that can be slewed to. The first character must be a 1 (channel 1). See '[SL' Start VFU Load, p.640](#)', '[Sn' Slew to Channel, p.637](#)', and '[VT' Slew to S6, Vertical Tab, p.645](#)'.

'EM' Mnemonic*End Output Markup Mode***Behaviour**

Description Use 'EM' to end output of markup files containing embedded mnemonics. This is the opposite of the **'BM'** Mnemonic, [p.591](#).

'EO' Mnemonic*End Output Transparency***Behaviour**

Description Use 'EO' to end output transparency mode. This is the opposite of the **'BO'** Mnemonic, [p.591](#). If you want to send  'EO' as a mnemonic, you must use it as a stand-alone mnemonic; e.g., PRINT 'BO', X\$, 'EO'.

To send  'EO' as part of your data, embed it in an expression (e.g., PRINT 'BO'+X\$+'EO') instead of using it as a stand-alone. Then, ProvideX evaluates 'EO' as an embedded  'EO' and sends it without interpretation to the device / printer.

'EP' Mnemonic*Start Expanded Print***Graphical Display/Printer or Character Printer**

Description Use 'EP' to begin expanded print (either double wide or double high depending on the type of printer). To end expanded print, use the **'EF'** Mnemonic, [p.603](#). On GUI devices, the 'EP' mnemonic will start output of standard text using a double-sized font and reset at the end-of-line.

'ER' Mnemonic*End Reverse Video***GUI Display or Character Display/Printer**

Description *Default.* Use 'ER' to end reverse video mode. This is the opposite of the **'BR'** Mnemonic, [p.592](#).

'ES' Mnemonic*Send Escape***Character Display/Printer**

Description Use the 'ES' mnemonic to send a `ESC` escape code to a text mode device.

'ET' Mnemonic*End Type Ahead***Editing or Behaviour**

Description Use 'ET' to end type-ahead mode. This is the opposite of the **'BT'** Mnemonic, [p.593](#).

'EU' Mnemonic*End Underscoring***Graphical Display/Printer or Character Display/Printer**

Description Use 'EU' to end the underscoring of characters. This is the opposite of the **'BU'** Mnemonic, [p.593](#).

'EW' Mnemonic*End WrapAround***Behaviour or Editing**

Description Use 'EW' to end WrapAround. (See **'BW'** or **'WRAP' WrapAround On/Off**, [p.648](#).)

'+F' & '-F' Mnemonics*Signal Change of Focus On/Off***Behaviour or GUI Display**

Format *Signal Change of Focus: '+F'*
Do Not Signal Change of Focus: '-F'

Description Use **'+F' & '-F'** to control whether or not ProvideX signals a change of Focus. If you use **'+F'**, ProvideX signals a change of Focus by issuing a `CTL=1000`.

'Fn' Mnemonic

Foreground Colour

Graphical Display/Printer or Character Display

*Where:**n* Numeric colour code. Default is 'F7'. Supported options are:

0 Black*	4 Blue	8 Dark Gray	< Light Blue
1 Red	5 Magenta	9 Light Red	= Light Magenta
2 Green	6 Cyan	: Light Green	> Light Cyan
3 Yellow	7 White	; Light Yellow	? Light Gray

Description Use the format above to begin outputting a foreground colour to a device (i.e., **PRINT**, **INPUT**, etc). The 16 colour codes are in ASCII sequence from 0 to ?.

Use the same colour codes for '**Bn**' **Background Colour**, p.588.

Example In this example, the prompt and user's response will be in white text on a blue background.

```
0010 INPUT 'B4', 'F7', "Please enter your name: ", NAME$,
```

'FF' Mnemonic

Form Feed

Graphical Display/Printer or Character Display/Printer

Description 'FF' is normally used with printers (as a form feed to advance a page) but you can also use it as an alternative to 'CS' for clearing the screen (to allow programs intended for printer output to send data to a display device). This is predefined to \$OC\$.

'FILL' Mnemonic

Define Fill Style

Graphical Display/Printer

Format

1. *One-Colour Fill Pattern*: **FILL'** (*pattern,colour1*)
2. *Two-Colour Fill Pattern/Gradient*: **'FILL'** (*pattern,colour1,colour2*)

*Where:**pattern* Numeric code for *fill pattern* type. Supported options include:

0 - No fill	4 - Crossed line
1 - Solid fill	5 - Bottom left to top right
2 - Horizontal lines	6 - Top left to bottom right
3 - Vertical lines	7 - Diagonal crossed lines

Numeric code for *gradient pattern* direction. Supported options include:

- 2 - top to bottom
- 3 - left to right
- 5 - top-left to bottom-right
- 6 - bottom-left to top-right

colour1 Fill colour. String or numeric. Use colour code, colour name, or RGB setting; i.e., RGB: *n n n* where *n*=0–255. Options include:

- | | |
|-------------------|-------------------|
| 0 - Black | 8 - Dark Gray |
| 1 - Light Red | 9 - Dark Red |
| 2 - Light Green | 10 - Dark green |
| 3 - Light Yellow | 11 - Dark Yellow |
| 4 - Light Blue | 12 - Dark Blue |
| 5 - Light Magenta | 13 - Dark Magenta |
| 6 - Light Cyan | 14 - Dark Cyan |
| 7 - White | 15 - Gray |

colour2 Second colour for *two-colour gradient* or *fill* patterns. String or numeric. Options are the same as for **colour1**; however, when applying two colours, ensure that both are defined using the same convention – if **colour1** uses RGB, **colour2** must use RGB as well.

Description Use 'FILL' to define the current fill pattern/gradient and colours for an open channel (default is the terminal). The direction for *gradient fill* is derived from the pattern codes 2, 3, 5, and 6. Non-gradient *two-colour fill patterns*: 4 and 7 (first colour for lines, second colour for background fill).

Examples

```
PRINT 'PEN' (1,3,1), 'FILL' (2,6), 'CIRCLE' (224,450,100)
PRINT 'Fill' (1, "RGB: 192,192,192"),
PRINT 'Fill' (0, "Light Red"),
PRINT 'Fill' (3, "Colur32", "Colur51")
```

'FL' Mnemonic

Start Function Key Load

Editing or Behaviour

Description Use 'FL' to start loading Function keys. This feature is included for compatibility with other languages. For more information on conversion and compatibility modes, see the *Installation Guide*. Also, refer to the [DEFCTL Directive, p.79](#).

ProvideX Utilities:

ProvideX utilities do not expect function and editing keys to be loaded with other values from the use of 'FL' or 'EL'. Issue a PRINT 'FL', "1", 'EL', "1" to reset loaded function or editing keys just prior to running any of the ProvideX utilities. See also: ['EL' Start Edit Key Load, p.604](#).

'FONT' Mnemonic

Define / List Fonts

Graphical Display/Printer

Formats

1. Define Font (with specification): **'FONT'(name\$,size[,attrib\$[,angle]])**
2. Define Font (via special names): **'FONT'({"*SYSFONT" | "*GUIFONT"})**
3. List Channel's Fonts: **'FONT'(LIST*[,chan])**
4. List Fonts, Properties: **'FONT'(LIST PROPERTIES FOR [name\$][,chan])**
5. List Sizes for a Font: **'FONT'(LIST name\$[,chan])**

Where:

angle Slant for printing, in degrees. Optional.

attrib\$ Font attributes. Optional string expression. Valid codes include:

- &** Underscore the character following the '&' (as in hot keys)
- B** Bold
- C** Centre text
- F** Show focus lines around text
- I** Italics
- N** Numeric data alignment
- R** Right justify text
- S** Applies background colour to area directly behind text.
- U** Underscore ("_")
- W** Word wrap
- #** Same as **N**

In addition, you can control the character set.

- !** Use symbol character set (char. set 2)
- O** Use OEM character set (char. set 255)
- J** Use Japanese character set (char. set 128)
- D** Use character set 1 or current default for the given font.
- %nnn** Use specific Windows character set *nnn*.
The default is ANSI, character set 0.

chan Logical file number or channel.

***GUIFONT** Keyword representing the dialogue font used by standard MS Windows applications for dialogues.

name\$ Font name. String expression. Font must exist in the system.

size Numeric. Use positive values for font sizes relative to the current default for the device (.5 for half size, 2 for double, etc.). Use negative sizes for absolute font size in points. As a rough guide to equivalent sizes:

- Point size -12 = 6 LPI, 10 CPI
- Point size -10 = 7.2 LPI, 12 CPI
- Point size - 7 = 10 LPI, 16 CPI.

***SYSFONT** Keyword representing the default graphical system font - typically "System, 0.66, B".

Description **'FONT'** defines the current font and specifications. It can also be used to return comma-delimited font and property (attribute) lists for the channel (default=the terminal), or a size list for a specific (existing) font.

Two special font names may be used in place of the specifications: ***SYSFONT** (default graphical system font) and ***GUIFONT** (standard MS Windows dialogue font). These fonts will change based on OS Version and theme. The new special font names may be used anywhere a font specification is given.

Example

```
f$="MS Serif"; ? 'FONT'(LIST F$) ! returns sizes for MS Serif font:
13,16,19,21,27,35,10,11,
0100 PRINT 'FONT'("MS Serif",-11)
```



Note: **'FONT'** does not work with direct-to-device printers; i.e., it can be used with ***WINPRT*** but not ***WINDEV***. See ***WINPRT*** [Windows Printing, p.760](#).

'FRAME' Mnemonic Define / Draw a Frame

GUI Display

Format **'FRAME'**(*x,y,x,y,style*,[*title\$*])

Where:

title\$ Frame title. Optional. String expression.

style Determines the style level and bevel sizes in creating the frame.
 Numeric expression:
 >0 **'FRAME'** is elevated button with bevel width=style value
 <0 **'FRAME'** is inset with bevel width=style value
 =0 **'FRAME'** appears etched or flat (depends on **'2D'**, **'3D'**, or **'4D'**).

x,y Placement coordinates, top left and bottom right, in graphical units. Numeric expression.

Description Use **'FRAME'** to draw (print) a frame on the screen. Use graphical units or **@X(col)** and **@Y(line)** functions for beginning and ending the frame.

Example












```
0010 PRINT 'FRAME'(100,100,450,450,10,"Hello There")
```

'GD' Mnemonic

Define Graphics Character Set

Definition

Description Use the 'GD' mnemonic to define the 11 characters that can be used for text mode *line-drawing* operations. The standard line-drawing characters are A-K (and for compatibility 0-9 and colon) as defined below:

A or 2		Top left corner.	G or :		Cross hairs.
B or 3		Top right corner.	H or 6		Vertical with horizontal right.
C or 4		Bottom left corner.	I or 7		Vertical with horizontal left.
D or 5		Bottom right corner.	J or 8		Horizontal with vertical up.
E or 0		Horizontal line.	K or 9		Horizontal with vertical down.
F or 1		Vertical line.			

Each character consists of up to four lines (each line represented by a bit in the byte defined by 'GD'); i.e.,

\$01\$ - Horizontal line centered top to bottom in left half of cell.

\$02\$ - Vertical line centered left-right in top half of cell.

\$04\$ - Horizontal line centered top to bottom in right half of cell.

\$08\$ - Vertical line centered left-right in bottom half of cell.

If 'GD' is not defined for the output, then \$0C090603050A0F0E0B070D\$ is the default. This conforms to the standard graphical character outputs as defined in the table above. The mnemonics 'GS' and 'GE' will not (and should not) be defined. If an output character is not defined by the 'GD' mnemonic, the cross hairs will be used.

For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

Examples

MNEMONIC(FIL_NO)'GD'=\$C9BBC8BCCDBACECCB9CACB\$! For double-line graphics

MNEMONIC(FIL_NO)'GD'=\$DABFC0D9C4B3C5C3B4C1C2\$! For single-line graphics

'GE' Mnemonic

End Graphics Data

Graphical Display/Printer or Character Display

Description Use 'GE' to end graphics data transmission. This is the opposite of the 'GS' Mnemonic, [p.613](#).

'GF' Mnemonic*Default Font for Window Objects***GUI Display**

Description Use **'GF'** to declare the current font in use as the default graphic font for all objects to be created in the window.

'GOTO' or 'WG' Mnemonic*Make Window Current***GUI Display or Character Display**

Format *Long or short form: 'GOTO' or 'WG'*
'GOTO'(wdw_id)

Where:

wdw_id Window's unique ID number (0 - 255).

Description Use either **'GOTO'** or **'WG'** in the format to make the identified window the current window and move it the top of the window stack. If this window does not exist or is the only window, ProvideX returns an Error #57: No such window defined.

Example

```
30400 LET WW_ADD=HWN(0)
30410 PRINT %W_MSG$, 'WINDOW' (20,8,40,8,WW_ADD,"Deleting Sku"
30410:), 'CS', 'SB',
30420 LET DEL_SKU$=FROM_SKU$, SAVE_KEY$=COMP$+FORM$
30430 !
30440 LET ADD_SKU$=TO_SKU$
30460 GOSUB DELETE_ITEM
30470 READ (E855FENT_H,KEY=SAVE_KEY$) IOL=0310
30480 PRINT 'WG' (W_ADD), 'WD' (WW_ADD), %NORM_SCR$,
```

'GREEN' & '_GREEN' Mnemonics*Colour Text***Graphical Display/Printer or Character Display**

Format *Foreground: 'GREEN'*
Background: '_GREEN'

Description All input or output following this mnemonic will be in green foreground or background.

Example

```
INPUT '_GREEN', 'WHITE', "Please enter your name: ", Name$,
```

'GS' Mnemonic*Start Graphics Data Transmission***Graphical Display/Printer or Character Display**

Description Use 'GS' to begin printing/displaying of (line-drawing) graphics. For a list of available line-drawing characters, refer to the mnemonic '**GD** Define Graphics Character Set, p.611'. To end transmission, use '**GE** End Graphics Data, p.611'.

'*H' Mnemonic*Control Screen Colours***Definition**

Format `'*H' =colour_codes$`

Where:

`colour_codes$` String of 8 characters representing screen colours for program listings. The following ASCII colour codes are supported:

0 Black*	4 Blue	8 Dark Gray	< Dark Blue
1 Red	5 Magenta	9 Dark Red	= Dark Magenta
2 Green	6 Cyan	: Dark Green	> Dark Cyan
3 Yellow	7 White	; Dark Yellow	? Dark Gray

Each colour code position represents different elements:

1: Background colour for highlighting `*[...]` searches

0 to **7** for standard background colours

8 to **?** for bright/foreground colours

R for Reverse Video.

2: Colour for variables.

3: Colour for literals.

4: Colour for remarks.

5: Colour for error lines.

6: Colour for mnemonics.

7: Colour for statement numbers or labels.

8: Colour for operators (e.g., `+ - () * /`).

For positions 2 to 8 the colour codes are **0** to **7** for standard foreground colours and **8** to **?** for dim / background colours. The command mode scanning feature uses Highlight=Yellow.

Description Use '***H**' (*Star-h*) to define colours in a displayed listing for the **LIST Directive, p.176**, and **LST() Function, p.477**. (See also: '**CS** System Parameter, p.660.) Default settings are shown in the example below.

Example `MNEMONIC '*H'=" ;4:>1=;9"`

'HIDE' Mnemonic*Control Window Display***GUI Display**

Description Same as **'SHOW'** / **'HIDE'** [Control Window Display, p.639](#).

'*I' Mnemonic*Input Conversion Table***Definition**

Format **'*I'** =table\$

Description **'*I'** (*star-i*) is used to create a 256-character terminal input conversion table. As each character is received from an input device, it is translated into a new character based on this table, if defined. An incoming character is translated to its numeric value in an ASCII table (0-255) and this value is used as an *offset* into the 256-character table defined by **'*I'** — the *character at that offset* will be used for input instead of the original incoming character.

For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

'+I' & '-I' Mnemonics*Implied Decimals On/Off***GUI Display or Behaviour**

Format *Enable Implied Decimals for Numeric Multi-lines: '+I'*
Disable Implied Decimals for Numeric Multi-lines: '-I'

Description Use PRINT '+I ' during system startup to activate support for implied decimal points for all numeric multi-lines. Use PRINT '-I ' to cancel it. (For a single multi-lines use OPT= 'i ' to suppress implied decimal use.)

'IC' Mnemonic*Insert a Space at Cursor***Editing or Behaviour**

Description Use 'IC' to insert a space at the current cursor position and shift all remaining characters on the line one position to the right.

'IMAGE' Mnemonic*Define a Graphics Group***Graphical Display/Printer****Formats**

1. Define Graphics Group: **'IMAGE'** (group\$)
2. Delete Graphics Group: **'IMAGE'**(DELETE group\$)
3. Enable/Disable Group: **'IMAGE'**({ENABLE | DISABLE} group\$)

Where:

group\$ Graphics group name. String expression. If you omit this when defining a group, the default is "" (null).

Description

ProvideX normally queues all mnemonics since the last **'CS'** (to supply information in case Windows needs to redraw the screen). In consequence, if you are constantly creating graphics, you can exhaust system resources. **'IMAGE'** helps to circumvent this potential problem.

When **'IMAGE'** is used to define and control the display of graphics groups, all graphical mnemonics following the **'IMAGE'** mnemonic will be grouped. You can then disable or hide the group to remove it from view and enable or show the group to make it visible.

The **DELETE** option can be used to destroy the group and return resources to the system.



Hint: When drawing graphical objects, each object will be laid one on top of the next. Use the **'IMAGE'** mnemonic to erase previous graphics instead — this reduces the consumption of Windows resources and helps prevent flicker during a repaint operation.

Example

```
0010 ! Display 'image' groups
0020 PRINT 'CS'
0030 LET FACE$='PEN'(1,4,3)+'ARC'(800,200,105,1,1,360)
0040 LET NOSE$='PEN'(1,4,0)+'LINE'(800,205,800,235)
0050 LET LEFT_EYE$='PEN'(1,2,4)+'FILL'(1,0)+'CIRCLE'(765,170,20)
0060 LET RIGHT_EYE$='PEN'(1,1,0)+'FILL'(1,0)+'CIRCLE'(835,170,18,1)
0070 LET WINK$='PEN'(1,4,0)+'ARC'(765,170,20,1,165,15)+'FILL'(1,0)+'CIRCLE'(775
0070: ,170,10)
0080 LET SMILE$='PEN'(1,3,0)+'ARC'(800,230,45,1,190,350)
0090 PRINT 'IMAGE'("HAPPY"),FACE$,NOSE$,SMILE$
0100 PRINT 'IMAGE'("WIDE_EYED"),LEFT_EYE$+RIGHT_EYE$; WAIT .5
0110 PRINT 'IMAGE'(DISABLE "WIDE_EYED")
0120 PRINT 'IMAGE'("WINK"),WINK$,RIGHT_EYE$; WAIT 1; PRINT 'IMAGE'(DELETE "WINK")
0130 PRINT 'IMAGE'(ENABLE "WIDE_EYED"); WAIT 1
0140 PRINT 'IMAGE'(DELETE "HAPPY"),'IMAGE'(DELETE "WIDE_EYED")
0150 PRINT @(0,12); LIST
0160 PRINT WINK$,RIGHT_EYE$,NOSE$,SMILE$; END
```

'JC' Mnemonic*Justify Centre***Graphical Display/Printer**

Description Use 'JC' to indicate that the following text is to be centre-justified.

'JD' Mnemonic*Justify Decimal-Aligned***Graphical Display/Printer**

Description Use 'JD' to indicate that the following data is to be decimal-aligned.

'JL' Mnemonic*Left-Justify Text***Graphical Display/Printer**

Description Use 'JL' to indicate that the following text is to be left-justified.

'JN' Mnemonic*Right-Justify for Numeric***Graphical Display/Printer**

Description Use 'JN' to indicate that the following numeric data is to be right-justified.

'JR' Mnemonic*Right-Justify Text***Graphical Display/Printer**

Description Use 'JR' to indicate that the following text is to be right-justified.

'JS' Mnemonic*Left-Justify String***Graphical Display/Printer**

Description Use 'JS' to indicate that the following string data is to be left-justified.

'L6' *Mnemonic**Set to 6 LPI***Graphical Printer or Character Printer**

Description On GUI devices, the **'L6'** mnemonic is equivalent to setting `'LPI' (6)`. On text mode devices, it is not predefined. See **'LPI' Logical Lines / Inch**, p.618.

'L8' *Mnemonic**Set to 8 LPI***Graphical Printer or Character Printer**

Description On GUI devices, the **'L8'** mnemonic is equivalent to setting `'LPI' (8)`. On text mode devices, it is not predefined. See **'LPI' Logical Lines / Inch**, p.618.

'LC' *Mnemonic**Mixed-Case User Input***Editing or Behaviour**

Description Allow upper and lower case (i.e., mixed case) for user input. This is the opposite of the **'UC' Mnemonic**, p.644.

'LD' *Mnemonic**Delete Current Line***Editing**

Description Delete the current line and shift all subsequent lines up one line.

'LF' *Mnemonic**Line Feed (Advance Line)***Graphical Display/Printer or Character Display/Printer**

Description Advance one line, while remaining in the same column position. (This can be redefined in the device driver, which may result in different behaviour.)

'LI' *Mnemonic**Insert Line***Editing**

Description Insert a blank line at the current position and push all subsequent lines down by one. The bottom line on a full screen will be lost.

'LINE' Mnemonic*Define / Draw a Line***Graphical Display/Printer**

Format **'LINE'**(*x,y,x,y[,x,y ...]*)

Where:

x,y Sets of *x,y* coordinates in graphical units. Numeric expression.

Description Use **'LINE'** to draw (print) a line on the device (e.g., terminal). Use graphical units or **@X(*col*)** and **@Y(*line*)** functions for the various coordinates. The **'LINE'** mnemonic draws these lines using the current **'PEN'** attributes.

Example 0010 print 'line' (224,450,355,420,210,400)

'LM' Mnemonic*Landscape Mode***Graphical Display/Printer**

Description Use **'LM'** to switch to landscape mode when printing to ***WINPRT***, *p.760*. This is the opposite of the **'PM'** Mnemonic, *p.633*.

'LPI' Mnemonic*Logical Lines / Inch***Graphical Printer**

Format **'LPI'**(*lines*)

Where:

lines Logical lines/rows per inch for graphics mode. Numeric expression.

Description Use **'LPI'** to set the logical LPI (lines per inch) value in graphics printing, where (as a rough guide to equivalent sizes):

Point size -12 = 6 LPI, 10 CPI

Point size -10 = 7.2 LPI, 12 CPI

Point size -7 = 10 LPI, 16 CPI

Refer to the **'CPI' Logical Characters per Inch**, *p.597*, and the functions **TXH()**, *p.544* and **TXW()**, *p.545*.

Example 0060 PRINT (30) 'LPI' (6),
0070 PRINT (1) 'LPI' (76/10),

'LT' Mnemonic*Move Left One Column***Motion**

Description Use 'LT' to move the cursor one column to the left.

Example

```
0010 PRINT "OH_X", 'LT', "Hello", 'RT', "There"
-:RUN
OH_Hello There
```

'MAGENTA' & '_MAGENTA' Mnemonics*Colour Text***Graphical Display/Printer or Character Display**

Format

Foreground: 'MAGENTA'
Background: '_MAGENTA'

Description All input or output following this mnemonic will be in magenta foreground or background.

Example

```
INPUT '_MAGENTA', 'WHITE', "Please enter your name: ", Name$,
```

'MAXSIZE' & 'MINSIZE' Mnemonics*Window Resize Limit***GUI Display**

Format

User's Maximum Setting: 'MAXSIZE'(width,height)
User's Minimum Setting: 'MINSIZE'(width,height)
Where:

width,height Size limits on the user's permission to resize a window. Width in columns, height in lines. Numeric expression.

Description Use 'MAXSIZE' and 'MINSIZE' to limit the maximum and minimum a window can be resized by the user when dragging the window's edge. These mnemonics only affect the window size that the user can set, not what the program sets as the size. They have no effect on the 'SIZE' mnemonic.



Note: The values you use for 'MINSIZE' must not exceed the values for 'MAXSIZE'. By default, 'MINSIZE' is set to 0,0 and 'MAXSIZE' is set to the originally-defined window size. If you set 'MAXSIZE' (0 , 0), then the total defined window size is used.

Example

In the example below, the user is limited to resizing the scrollable dialogues display area by reducing it to a minimum of 5 columns by 5 lines or increasing it a maximum of 40 columns by 15 lines:

```
0010 PRINT 'DIALOGUE'(1,1,80,25,"Title",'CS',OPT="Z")
0020 PRINT 'SIZE'(30,10),
0030 PRINT 'MINSIZE'(5,5),
0040 PRINT 'MAXSIZE'(40,15),
```

'ME' Mnemonic*Begin Edit Mode***Editing or Behaviour****Description**

Use **'ME'** to begin Edit Mode. In Edit Mode, any keystroke or negative **CTL** event not used by the input (e.g., up / down arrows) will be returned to your program in the **CTL** variable instead of being rejected. ProvideX uses built-in edit functions to format and process all keyboard input, but will terminate input and returns values to your program when it encounters keystrokes, or negative **CTL** events the input editor doesn't handle. To end Edit Mode, use the **'MN'** Mnemonic, [p.622](#).

Example

```
0010 INPUT "Enter Amount:", 'ME', AMNT:"$###,##0", 'MN'
```

'MESSAGE' Mnemonic*Define Message Bar Text***GUI Display****Format**

1. *Display Text in Message Bar:* **'MESSAGE'**(text\$)
2. *Define Segmented Message:* **'MESSAGE'**(DEF start_1[,start_2[,start_3]])
3. *Reset Message Bar:* **'MESSAGE'**(DEF 0)
4. *Display Text in Specific Segment:* **'MESSAGE'**(text\$,segment)

ext\$ Text to display. String expression.

start_# Column number where optional additional segments begin. Default segment 0 starts at column 0. You can have up to 4 segments (0, 1, 2 and 3). Define optional segments 1, 2 and 3 by specifying their starting column number. Numeric expressions.

segment Segment # for display. Numeric expression. Valid range: 0 to 3.

Description

Use **'MESSAGE'** to print text on the message bar at the bottom of the ProvideX window. When you can create optional message bar segments 1, 2 and 3, ProvideX places the segment separator at the column number you specify in **start_#** for the corresponding segment(s). Reset to a single segment (segment 0 at column 0) by Defining segment 0 *zero*.

You can send **'MESSAGE'** to segment 0 *zero* and to defined segments by segment number. By default, if you omit the segment number, your text is displayed in segment 0 starting at column 0.



Note: If you use a positive column number, the segment's separator is offset that many columns from the left of the message bar. Use negative values to have the separator's placement offset from the right instead. If you want to centre text within a segment, use \$09\$ (**Tab** character) as the first character of the text to print.

Example

```
0010 print 'message'(def 7,-20) ! create seg1 @(7), seg2 @(20 cols from right)
0020 print 'message'("") ! Null displayed in segment 0
0030 print 'message'("hello",1), 'message'("there",2)
0040 print "To reset the message bar PRINT 'message'(def 0)"
```

Message Bar Region Events

LEFT-MOUSE-CLICK and RIGHT-MOUSE-CLICK events are now supported in the message bar region. ProvideX returns CTL values when the user clicks on a segment in the **'MESSAGE'** region.



Note: The existence/height of the message bar can be controlled by an INI file setting.

Each of the four possible segments of the message bar region has been assigned a different negative CTL value. The event is reported on the button UP only. The return values are shown in the chart below.

'MESSAGE' Region	LEFT-MOUSE-BUTTON-UP	RIGHT-MOUSE-BUTTON-UP
1st area (segment zero)	CTL= -1400	CTL= -1410
2nd area (segment 1)	CTL= -1401	CTL= -1411
3rd area (segment 2)	CTL= -1402	CTL= -1412
4th area (segment 3)	CTL= -1403	CTL= -1413

'MINSIZE' Mnemonic

Window Resize User Limit

GUI Display

Description Same as **'MAXSIZE' & 'MINSIZE' Window Resize User Limit**, p.619.



'MN' Mnemonic

End Edit Mode

Editing or Behaviour

Description Use **'MN'** to end Edit Mode. This is the opposite of the **'ME'** Mnemonic, *p.620*. After use of the **'MN'** mnemonic, all keystrokes or negative **CTL** events not used by the input will be thrown away, and *not* returned to the program.

Example `0090 OBTAIN (0,SIZ=1,ERR=0090)@(0,0), 'CURSOR'("off"), 'ME', IN_VAR$, 'MN'`

'MODE' Mnemonic

Set Attributes and Colour

Character Display

Format `'MODE'(attrib[$])`

Where:

`attrib[$]` Attribute represented by a string or numeric value.

Description Use **'MODE'** to directly reset the current attributes and colour of text in a character-based display.



Note: This mnemonic is supported for backwards compatibility of legacy code. For graphical applications, use the mnemonics listed under **GUI Display**, *p.583*.

A one-, two- or three-character attribute string is represented as follows:

- 1 character changes the attribute only.
- 2 characters change the foreground & background colours only.
- 3 characters change the attribute (represented by first character) as well as the foreground & background colours (represented by second and third characters).

If a numeric value is provided, the low order 8 bits is considered as a single character string and is assumed to contain the new attribute setting. If either of the colour bytes contain the value `FF` then the colour byte is ignored and the current colour (foreground or background) remains as is.

The attribute byte is defined as follows:

- `01` Background
- `02` Inverse Video
- `04` Blinking
- `08` Underscore
- `10` Graphic character



'MOVE' or 'WM' Mnemonic *Relocate Current Window*

GUI Display or Character Display

Format *Long or short form: 'MOVE'(col,line) or 'WM'(col,line)*
Where:
col,line Starting coordinates (top left) of new location. Numeric expressions.

Description Use either 'MOVE' or 'WM' to relocate the current window to a new placement starting with the top left corner at the *col,line* (column and line) coordinates.

'MP' Mnemonic *Print Mode (Parallel)*

Graphical Display/Printer or Character Display/Printer

Format 'MP'

Description Use 'MP' to switch the printer from serial printer mode to line printer (parallel) mode. In line printer mode, overstrike data will print on top of existing characters at their positions on the print line. Use 'MS' to switch the printer to serial mode. See also 'SP' System Parameter, p.687.

'MS' Mnemonic *Print Mode (Serial)*

Graphical Display/Printer or Character Display/Printer

Format 'MS'

Description Use 'MS' to switch the printer from line printer (parallel) mode to serial printer mode. In serial printer mode, overstrike data will replace existing characters on the print line. Use 'MP' to switch the printer to line printer (parallel) mode. See also 'SP' System Parameter, p.687.

'+N' & '-N' Mnemonics *Control Drop/List Box Write Error*

Behaviour or GUI Display

Format *Do not return Error 11: '+N'*
Return Error 11: '-N'

Description Use '-N' to generate an `Error 11` on a `DROP_BOX WRITE " "` or `LIST_BOX WRITE " "` if no such entry exists. '+N' tells the system to clear any selected item when writing a null string. These mnemonics have no effect on multi-select list boxes.

'NI' Mnemonic

Next Input Numeric

Editing

Description Use 'NI' to indicate that the next input is to contain only numeric data (0-9, -, \$, the comma, and the decimal point).

'*O' Mnemonic

Output Conversion Table

Definition

Format `*O'=table$`

Description `*O'` (*star-o*) is used to create a 256-character terminal output conversion table. As each character is sent to an output device, it is translated into a new character based on this table, if defined. The outgoing character is translated to its numeric value in an ASCII table (0-255) and this value is used as an *offset* into the 256-character table defined by `*O'` — the *character at that offset* will be used for output instead of the original character.

For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

'OPTION' Mnemonic

On-The-Fly Setting

Behaviour

Format `'OPTION' (keyword$,value$)`

Where:

keyword\$ Named setting. Case-insensitive string expression (characteristic, icon, font, colour index) described below.

value\$ Value assigned to *keyword\$*. String expression.



Description

The 'OPTION' mnemonic allows various ProvideX environment settings, (typically set via INI entries) to be changed on the fly at run time within a ProvideX session. These include [Item Colour Settings](#), [Item Shading Settings](#), [Icon Settings](#), [Font Settings](#), [Resource Library](#), [Colour Index](#), [PDF Bookmarks](#), [Debugging Functionality](#), [*WINPRT* Printing Options](#).



Note: Several other 'OPTION' settings are available specific to ProvideX thin-clients, such as JavX and UltraFX. These are documented in the ProvideX *Client-Server Reference*.

Item Colour Settings

The following *keyword*\$ settings assign colours to specific characteristics:

"BtnFocusHiligh	Focus highlight
"BtnHoverHiligh	Mouse Over highlight
"BtnFaceClr"	Button face colour
"BtnFrameClr"	Button/check box/radio button frame colour
"CbxMarkClr"	Check mark or 'X' in a check box
"RbtMarkClr"	Ball within a radio button
"FrameTextClr"	Text in a frame.

The associated *value*\$ may be the colour name Black, Light Red, Light Green, Light Yellow, Light Blue, Light Magenta, Light Cyan, White, Dark Gray, Dark Red, Dark Green, Dark Yellow, Dark Blue, Dark Magenta, Dark Cyan, Gray ... or the RGB code (i.e., RGB: *n n n* where *n*=0-255); e.g.,

```
PRINT(0) 'option' ( "BtnNormMidClr" , "RGB: 192,192,192" ) ,
```

Item Shading Settings

The following *keyword*\$ settings assign shading for specific characteristics:

"BtnNormTopPct"	Lightness applied to face at the top of normal button.
"BtnNormTopClr"	Colour for blend at the top of normal button.
"BtnNormMidPct"	Lightness applied to face in middle of normal button.
"BtnNormMidClr"	Colour for blend at in middle of normal button.
"BtnNormBtmPct"	Lightness applied to face at the bottom of normal button.
"BtnNormBtmClr"	Colour for blend at the bottom of normal button.
"BtnNormMiddle"	Mid point percentage from the top of normal button.
"BtnDownTopPct"	Lightness applied to face at the top of pushed button.
"BtnDownTopClr"	Colour for blend at the top of pushed button.
"BtnDownMidPct"	Lightness applied to face in middle of pushed button.
"BtnDownMidClr"	Colour for blend at in middle of pushed button.
"BtnDownBtmPct"	Lightness applied to face at the bottom of pushed button.
"BtnDownBtmClr"	Colour for blend at the bottom of pushed button.
"BtnDownMiddle"	Mid point percentage from the top of pushed button.
"FrameTextClr"	Colour of any text used in a frame.

"CtlFrameClr" Colour of lines in a frame.
 The associated *value\$* depends on the *keyword\$*, either 0 to 100 (for colours) or -100 to 100 (for percentages); e.g.,
 PRINT(0) 'option' ("BtnDownBtmPct" , "-75 ")

Icon Settings

The following *keyword\$* settings can be used to control which icon, if any, is to be used within the upper left corner of a window/dialogue:

"ICON" Set the icon for current and subsequent windows.
 "WDWICON" Set the icon for the current window only.

The associated *value\$* may be the path and name of the image .ico file, or a full icon specification. If a null value "" is specified for "WDWICON", then the icon is removed from the current window. If it is an *asterisk* *, the default icon is displayed for the window. For example,

```
PRINT 'dialogue'(1,1,80,25,"My Window",'cs',opt="i")
PRINT 'option'("WDWICON","C:\Windows\System32\Shell32.dll@137%32")
PRINT 'option'("WDWICON","")
PRINT 'option'("WDWICON","*")
```

Font Settings

The following *keyword\$* settings can be used to change various text plane and graphic fonts on the fly:

"FONT" Set the current window's text plane font.
 "STDFONT" Set the session's default text plane font.
 "GRAPHICFONT" Set the current window's default graphic font.
 "STDGRAPHICFONT" Set the session's default graphic font. If set, this must be prior to drawing any GUI controls.

The associated *value\$* indicates the current font specification. Refer to the '[FONT](#)' [Mnemonic, p.609](#) for specifications. If the numeric font size specified in *value\$* is negative, then it provides the logical font height *not* the point size.

Resource Library

Resource libraries are DLLs that may contain icons and bitmaps for use in an application. The following *keyword\$* can be used to change the current resource library:

"RESOURCELIB" Set the current resource library.

The associated *value\$* may be the path and name of the library .DLL file.

Colour Index

The following *keyword*\$ can be used to manually define specific colours (via RGB code) to be added to the internal colour index (palette):

"**COLOUR***nnn*" Set colour index number *nnn*.
 "**COLOR***nnn*" Set color index number *nnn*.

While the index can include from 0 to 254 colours, the first 16 are predefined by ProvideX (0 to 15). Existing index numbers may be re-assigned to a new RGB value; however, new colours must be assigned *in sequence* using the next available index number (i.e., COLOUR16, then COLOUR17, then COLOUR18 ...). Be aware that colours defined via the '**COLOUR**' mnemonic are added automatically to the next available index number if they don't already exist in the palette.

The associated *value*\$ defines the colour assigned to the index number. This may be any RGB code (RGB:*n n n* where *n*=0-255); e.g.,

```
PRINT 'OPTION' ("Colour22", "RGB: 200,200,200")
```

A null "", either removes the assigned number from the index or resets it to the default colour (if predefined by ProvideX); e.g.,

```
PRINT 'OPTION' ("Colour64", "")    Clear out index number 64 for re-use
PRINT 'OPTION' ("Colour7", "")    Reset to the ProvideX default
```



Note: Use **FIN**(0,"**COLOUR***nnn*") to retrieve current user-defined colour settings.

PDF Bookmarks

Bookmarks (used for selecting and automatically displaying specific pages) can be added to generated PDFs. The following *keyword*\$ sets the bookmark:

"BookMark" Set PDF bookmark.

The associated *value*\$ defines a bookmark's location, text, and hierarchy for a PDF. For details, see [Creating Bookmarks](#) under ***PDF* PDF Print Interface, p.744**.

Debugging Functionality

Debugging functionality includes the ability to add watch values, set dynamic breakpoints, initiate program tracing, as well as the ability to transfer the contents of the debug window to the clipboard. This can be set on-the-fly using the *keyword*\$ "DebugWindow", which determines the specific functionality based on the associated *value*\$ settings; e.g.,

```
PRINT 'OPTION' ("DebugWindow", "Trace"),
PRINT 'OPTION' ("DebugWindow", "Host Command Open"),
PRINT 'OPTION' ("DebugWindow", "Watch Add "x$"),
```

The available DebugWindow *value*\$ settings for enhanced debugging functionality are described by category in the sections below:

Break Window Specific

"Break" Activate Break window.

Command Window Specific

"Command" Activate Command mode window.

"Halt" Simulate a Command window halt (stop program).

"Step" Simulate a Command mode window step.

Trace Window Specific

"[Host] Log [All] [Errors] [Enable|Disable]"
 Enable logging of errors on the host or local machine.

"Size=1k" Set 1K program trace size.

"Size=2k" Set 2K program trace size.

"Size=8k" Set 8K program trace size.

"Size=16k" Set 16K program trace size.

"Size=32k" Set 32K program trace size.

"Suppress [Program [Trace [Enable|Disable]]]"
 Enable/disable suppress program trace option.

"Trace" Activate Trace window.

"Host Trace Programs [Enable|Disable]"
 Enable/disable host trace programs option.

"Show [Property] [GET] [Enable|Disable]"
 Enable/disable show property GET option.

"Show [Property] [SET] [Enable|Disable]"
 Enable/disable show property SET option.

"File Opens [Enable|Disable]" Enable/disable trace file opens option.

"File Opens [Failures] [Enable|Disable]"
 Enable/disable trace file open failures option.

"File [IO] [Operation] [Trace] [Enable|Disable]"
 Enable/disable file IO operation trace option.

"DebugPlus [with] [Backtrace] [Enable|Disable]"
 Enable/disable debugplus backtrace option.

Watch Window Specific

"Size=0" Select the no-data-break option.

"Size=50" Select the 50-byte-data-break option.

"Size=100" Select the 100-byte-data-break option.

"Size=no" Select the No-data-break option.

"Watch" Activate Watch Window.

Multi-Use Commands

"Host [Break|Trace|Watch] [Enable|Disable]"
 Enable/disable host trace option.

"[Host] [Break|Watch] Add "String to Add"

Add a (pre-formatted) item to the break|watch window.

When using the following commands, if the debug window type is not specified, then the last window used will be utilized:

"[Host] [Break|Trace|Watch] Clear" Clear window contents

"[Host] [Break|Command|Trace|Watch] Close" Close window option.

"[Host] [Break|Trace|Watch] Copy" Copy window contents to Clip_Board.

"[Host] [Break|Command|Trace|Watch] Disable" Disable window without closing.

WINPRT Printing Options

WINPRT functionality can be set on-the-fly using:

"Source" Changes the *source tray* when printing to *WINPRT*.

The associated *value\$* can contain any valid source tray number supported by the printer. It can also be the source tray *name* as identified by **FIN(chan,"SOURCELIST")**.

"Orientation" Changes *paper orientation* when printing to *WINPRT*.

The associated *value\$* can contain "P" or "Portrait", "L" or "Landscape" and is case-insensitive.

Both "Source" and "Orientation" options may be issued at any point on a page, but will not take affect until the start of a new page.

'OFFSET' Mnemonic

Offset for *WINPRT*

Graphical Printer

Format 'OFFSET'(x,y)

Where:

x,y Offset coordinates in thousandths of an inch. Numeric expression.

Description This is the 'OFFSET' property used for *WINPRT*. It allows the user to change the offset to the printable area from the upper left corner of the page. For more information on *WINPRT* properties, see the [WINPRT_SETUP Directive, p.376](#), and [*WINPRT* Windows Printing, p.760](#).

'+' & '-' Mnemonics**Define Mouse Movement****GUI Display**

Format *Signal Pixel-to-Pixel Change: '+'*
 Signal Line/Column Change: '-'

Description Use '+' to transmit mouse movements on any change of position (pixel by pixel movement). Use '-' to transmit mouse movements only on a change of line / column.

'PE' Mnemonic**Auxiliary Port Off****Character Printer**

Description 'PE' contains the auxiliary port *off sequence* that tells the terminal to stop sending output to the attached printer and to send output back to the terminal's display area. This should be defined in the device driver for the terminal type then retrieved and set on the printer channel after opening the printer. To set the auxiliary port *on sequence* use the **'PS' Mnemonic, p.634**.

'PEN' Mnemonic**Define Pen Style****Graphical Display/Printer**

Format 'PEN'(style,width,colour)

Where:

colour 'PEN' colour. Use colour code, colour name, or RGB setting; i.e.,

RGB: *n n n* where *n*=0-255. Options include:

0 - Black	8 - Dark Gray
1 - Light Red	9 - Dark Red
2 - Light Green	10 - Dark green
3 - Light Yellow	11 - Dark Yellow
4 - Light Blue	12 - Dark Blue
5 - Light Magenta	13 - Dark Magenta
6 - Light Cyan	14 - Dark Cyan
7 - White	15 - Gray

style Numeric code for fill pattern type. Supported options include:

0 - No Pen	3 - Dotted
1 - Solid Pen	4 - Dash-dot
2 - Dashed	5 - Dash-dot-dot

width Pen width, in graphical units. Numeric expression



Note: 'PEN' styles 2 (Dashed), 3 (Dotted), 4 (Dash-dot) and 5 (Dash-dot-dot) only work if *width* is 1. 'PEN' style 1 (Solid) is the only style that works if *width* is greater than 1. (This is an internal Windows API specification/ restriction).

Description Use 'PEN' to define the current pen style, width, and colour for graphics drawing.

Example

```
'PEN' (1,10,6)
'PEN' (1,"RGB: 192,192,192")
'PEN' (0,"Light Red")
```

'PICTURE' Mnemonic

Define / Draw Picture

Graphical Display/Printer

Format

1. *Define Picture:* 'PICTURE'(x,y,x,y,{name\$|#chan[,transp_opt]}[,display_opt])
2. *List Embedded Pictures:* variable\$='PICTURE'(*)

Where:

* *Asterisk.* To have ProvideX return a list of its embedded pictures; e.g.,
X\$='PICTURE'(*)
PRINT X\$

This argument is not directly supported in WindX. Use syntax similar to the following to enable ProvideX on the server to retrieve a list of images from the workstation:

```
CALL "[WDX]*Windx.utl;get_val","'picture'(*)",x$
```


#chan String consisting of a # plus the channel containing the graphic; i.e, the channel opened via ***BITMAP***, p.738.

display_opt Numeric code—define display of graphic. Supported options include:

0=Align at top-left
1=Centre/crop within region
2=Scale to fit
3=Tile bitmaps to fill the given area
4=Halftone for enhanced legibility (may lighten black images)
5=Scale with proper aspect ratio but output in top left
6=Scale with proper aspect ratio but centred in the region.

For options 0, 2, and 3, the image is cropped to fit within the region for the screen and ***WINPRT*** output; however, no cropping is supported with ***PDF***. For option 3 (tiled) and PDF, only images that fit completely inside the region will be output (no cropping).

name\$ Name of graphic (e.g., C:\your_PATH\your.bmp). String expression. The icon filename must have .ico as an extension.

<code>x,y,x,y</code>	Point / position coordinates for top left and bottom right, in graphical units. Numeric expression.
<code>transp_opt</code>	Transparency options: G specifies that all colours of RGB value 192,192,192 (Light Gray) are considered to be transparent. T specifies that the colour of the first pixel in the upper left corner of the image is to be transparent.
	Note: The transparency options are not intended for <i>printed output</i> (i.e., *WINPRT* , *BITMAP* , *PDF*) and can produce <i>incorrect results</i> . Under UNIX/Linux, the use of these options with *PDF* will generate an Error #99: Feature not supported.

Description Use '**PICTURE**' to draw (print) a picture on the device (e.g., terminal). `x,y,x,y` coordinates define placement and size (top left and bottom right corners). Use graphical units or **@X(col)** and **@Y(line)** functions for the various coordinates. For displaying image transparency, place a G or T at the end of the image filename; i.e.,

```
PRINT 'PICTURE' (1,1,100,100,"myimage.bmp,T",0)
BUTTON 10,@(40,2,10,1.6)="{myicon.ico,T}&Name"
```

For internal images (i.e., those specified with an exclamation within braces { !imagename }), it is not necessary to use the G option because ProvideX always assumes this transparency on an internal bitmap, unless it is overridden with T. Transparent images are only supported when the picture does not need to be scaled. ProvideX cannot mask a scaled bitmap since the scaling process may alter colour codes.

'PIE' Mnemonic

Define / Draw Pie Slice

Graphical Display/Printer

Format '**PIE**(`x,y,radius,aspect,angle_1,angle_2`)

Where:

`angle_1` Starting angle, in degrees. Numeric expression.

`angle_2` Ending angle, in degrees. Numeric expression.

`aspect` Aspect ratio / viewpoint. (Ratio=1 results in no tilt.) Numeric expression.

`radius` Radius of the pie, in graphical units. Numeric expression.

`x,y` Coordinates for the centre of the pie, in graphical units. Numeric expression.

Description Use '**PIE**' to draw (print) a pie slice on the device (e.g., terminal). Use graphical units or **@X(col)** and **@Y(line)** functions for `x`, `y`, and `radius`. The pie slice extends from the starting `angle1` to `angle2`. The '**PIE**' mnemonic uses current attributes for the '**FILL**' Mnemonic, [p.607](#), and the '**PEN**' Mnemonic, [p.630](#).

Example The following example displays a pie cut into two uneven slices:

```
0030 PRINT 'PEN' (1,3,8), 'FILL' (2,6)
0040 PRINT 'PIE' (224,450,100,1,45,1)
0050 PRINT 'PEN' (1,3,6), 'FILL' (4,15)
0060 PRINT 'PIE' (260,440,100,1,1,45)
```

'PM' Mnemonic

Portrait Mode

Graphical Display/Printer

Description Use 'PM' to switch to portrait mode when printing to **WINPRT**, p.760. This is the opposite of the **'LM' Mnemonic**, p.618.

'POLYGON' Mnemonic

Define/Draw a Polygon

Graphical Display/Printer

Format 'POLYGON'(x,y,x,y,x,y,x,y ...)

Where:

x,y Set of point/position coordinates in graphical units. Numeric expression.

Description Use 'POLYGON' to draw (print) a polygon (e.g., triangle, hexagon ...). ProvideX joins the various x,y points to form the polygon. Use graphical units or @X(*col*) and @Y(*line*) functions for the coordinates.

The 'POLYGON' mnemonic uses current attributes for the **'FILL' Mnemonic**, p.607, and the **'PEN' Mnemonic**, p.630.

Example The example below creates an irregular four-sided figure by setting the coordinates for the four corners:

```
0030 PRINT 'PEN' (1,3,8), 'FILL' (2,6)
0040 PRINT 'POLYGON' (224,450,100,100,400,200,390,390)
```

'POP' or 'WR' Mnemonic

Restore Previous Window

GUI Display or Character Display

Format *Long or short form:* 'POP' or 'WR'

Description Use either 'POP' or 'WR' to remove the current window from the top of the stack and restore the previous window.

'PS' Mnemonic*Auxiliary Port On***Character Printer**

Description 'PS' contains the *on* sequence that tells the terminal to route incoming characters to the auxiliary port instead of the terminal's display area. This sequence remains active until an *off* sequence is encountered via the '**PE**' Mnemonic, p.630. This should be defined in the device driver for the terminal type then retrieved and set on the printer channel.

'PUSH' or 'WC' Mnemonic*Save/Copy Current Window***GUI Display or Character Display**

Format *Long or short form: 'PUSH' or 'WC'*

Description Use either '**PUSH**' or '**WC**' to save and copy the current window to create a new window with exactly the same size, position and attributes.

'*R' Mnemonic*OS Command String***Definition**

Format '*R'=command\$

Description '*R' (*star-r*) contains the operating system command that will be executed when the channel is closed and after all printing is completed.

For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

'RB' Mnemonic*Ring Bell***Editing**

Description Use '**RB**' to ring the bell.

'RC' Mnemonic

Return Cursor Address

Editing

Description Use **'RC'** to return the current cursor address as a four-character string containing the current line and column. The value will be returned in the input queue, just as a keystroke would, and may be read via a **READ RECORD**.

'RECTANGLE' Mnemonic

Draw a Rectangle

Graphical Display/Printer

Format **'RECTANGLE'**(*x,y,x,y,radius*)

Where:

x,y,x,y Coordinates for top left/bottom right corners, in graphical units.

radius Positive integer representing the *rounding factor* for corners. A negative value will have its sign flipped. If the radius exceeds half the height (or width), the system assumes a semi-circle is used to round the edge.

Description Use **'RECTANGLE'** to draw (print) a rectangle defined by two sets of x,y coordinates. Use graphical units or **@X(col)** and **@Y(line)** functions for the coordinates. The **'RECTANGLE'** mnemonic uses current attributes for the **'FILL'** Mnemonic, *p.607*, and the **'PEN'** Mnemonic, *p.630*.

Example

```
0030 PRINT 'PEN' (1,3,8), 'FILL' (2,6)
0040 PRINT 'RECTANGLE' (100,100,400,600)
0040 PRINT 'RECTANGLE' (700,100,820,220,30)
```

'RED' & '_RED' Mnemonics

Colour Text

Graphical Display/Printer or Character Display

Format *Foreground:* **'RED'**
Background: **'_RED'**

Description All input or output following this mnemonic will be in red foreground or background.

Example INPUT **'RED'**, **'_WHITE'**, "Please enter your name: ", Name\$,

'RL' Mnemonic*Return Line Contents*

Editing

Description Use **'RL'** to return the contents of the current line with the next terminal input.



Note: In graphics mode (**'GS'**), if a terminal mnemonic transmits the contents of the screen to the program, the data consists of four bytes for each character. The first two bytes are background and foreground characters, the third byte holds character attributes, and the fourth byte is the actual character on the screen.

Example

```
0040 PRINT 'CS'
0050 PRINT @(5,5),DIM(10,"A")
0060 INPUT @(5,5),'RL',B$
0070 PRINT B$
```

When Run:

```
AAAAAAAAAA
AAAAAAAAAA
```

'RM' Mnemonic*Reset to Default Mode*

Behaviour

Description Use **'RM'** to reset to default modes/attributes (colour, reverse video mode, underscoring mode, etc.). See also **'SN' Native Screen Mode, p.640** and **'SX' Set Extended Screen Mode, p.641**.

'RP' Mnemonic*Terminal Read to End*

Editing

Description This mnemonic performs the same function as **'TR' Terminal Read from Start, p.643**, except that it reports values from the current cursor location to the end of the screen. (**'TR'** returns the screen contents from 0,0 to the current cursor location.)

'RS' Mnemonic*Restore Screen*

Character Display or Editing

Description Use **'RS'** to restore the complete terminal screen from the information in memory. This mnemonic can be used to reset the screen after transmission errors or when operating system output has disrupted screen contents.

'RT' Mnemonic*Move Right One Column***Motion**

Description Use **'RT'** to move the cursor one column to the right.

Example

```
0010 PRINT "Oh_x" , 'LT' , "Hello" , 'RT' , "There"
-:RUN
Oh_Hello There
```

'+S' & '-S' Mnemonics*Substitute Solid Lines On/Off***Behaviour or Graphical Printer**

Format *Substitute Solid Lines: '+S'*
Do not Substitute: '-S'

Description When you use **'+S'**, ProvideX automatically replaces two or more occurrences of the underscore, dash or equals sign (_ - =) with solid underlines in graphics mode when printing to ***WINPRT***. This only applies to fields that are printed separately. (Primarily for use in legacy code where these characters were used in place of solid lines.) Use **'-S'** to disable **'+S'**.



Note: This is not for use on display devices.

Example

```
0010 DIM A$(10, "_"), B$(10, "-"), C$(10, "=")
0020 LET CHAN=UNT; OPEN (CHAN,ERR=*END)*winprt*
0030 PRINT (CHAN)'FONT'("MS Sans Serif",1), 'DF',
0040 PRINT (CHAN) '-S', @(0), A$, @(12), B$, @(24), C$
0050 PRINT (CHAN) '+S', @(0), A$, @(12), B$, @(24), C$
0060 PRINT (CHAN)@(0), A$+" "+B$+" "+C$
```

In the example above, ProvideX will only print solid lines when it executes line 0050. It will neither print solid lines for line 0040 (with '-S') nor for line 0060 (where the string expression does not exclusively contain underscores, dashes and/or equals signs).

'Sn' Mnemonic*Slew to Channel***Character Printer**

Description Use **'Sn'** to slew to the channel specified by *n*. **'S6'** is the same as **'VT'**. See **'EL' End VFU Load, p.604**, **'SL' Start VFU Load, p.640**, and **'VT' Slew to S6, Vertical Tab, p.645**.

'SB' Mnemonic*Set Mode to Background***Character Display**

Description Use **'SB'** to set background mode. (Characters are displayed at low intensity / dimmed.) To clear background mode, use the **'CF'** Mnemonic, p.595. (See also: **'SF'** *Set Mode to Foreground*, p.639.)

'SCROLL' Mnemonic*Define/Control Scroll Region***GUI Display or Character Display or Editing**

Format

1. *Define Region:* **'SCROLL'**(col,ln,wth,ht)
- Long or short forms:*
2. *Start / Enable Scrolling:* **'SCROLL'**("ON") or **'SE'**
3. *Reset to Full Window:* **'SCROLL'**("RESET") or **'SR'**
4. *Disable Scrolling:* **'SCROLL'**("OFF") or **'SD'**

Where:

@(col,ln,wth,ht) Position / coordinates. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.

Description Use **'SCROLL'** to define or change the scroll region in the screen / window. All subsequent mnemonics for cursor position, clearing, deletion and insertion will only affect this defined area.

If the cursor advances beyond the last line: when **'SCROLL'** is **ON**, ProvideX moves all the lines on the screen up one line; when **'SCROLL'** is **OFF**, ProvideX returns the cursor to the first column in the first line. Use **'SE'** or **'SD'** to enable or disable the scroll region. Use **'SR'** or **'SCROLL'**("RESET") to reset the scroll region to the full window / screen. For a window with a border, the border is included in the scroll region.



Note: The values **ON**, **OFF** and **RESET** can be literals, variables or string expressions.

'SE' & 'SD' Mnemonics*Scroll Enable/Disable***GUI Display or Character Display or Editing**

Description See **'SCROLL'** *Manipulate Scroll Region*, p.638.

'SF' Mnemonic*Set Mode to Foreground***Character Display**

Description Use **'SF'** to set foreground mode. (Characters are displayed at high intensity.) To clear foreground mode, use the **'CF'** Mnemonic, *p.595*. (See also: **'SB'** *Set Mode to Background*, *p.638*.)

'SHOW' / 'HIDE' Mnemonics*Control Window Display***GUI Display**

Format

1. *Show Window*: **'SHOW'(n)**
2. *Hide Window*: **'HIDE'**

Where:

n Numeric code. Supported options include:

- 0 = Minimizes current window
- 1 = Restores current window to normal display state
- 2 = Maximises current window
- 3 = Resizes current window to previous display state
- 1 = Hides the current window.

Description Show or hide a window (graphics display only). The default is to display the window.



Note: You must restore or show a hidden window through your program, since you can't send data input directly to it.

'SIZE' Mnemonic*Control Visual Size of Window***GUI Display**

Format **'SIZE'(width,height)**

Where:

width,height Window's width in columns and height in lines. Numeric expression.

Description Use **'SIZE'** to control the visual size of a window.



Note: There is no effect on the **'SIZE'** mnemonic when you use **'MAXSIZE'** & **'MINSIZE'** to control the window size that the user can set.

Example To resize a dynamic window to the columns and lines specified:

```
PRINT 'SIZE' (40,10)
```

'SL' Mnemonic

Start VFU Load

Character Printer

Description Use **'SL'** to start VFU load. The data following **'SL'** (from **'SL'** up to an **'EL'**) defines the VFU channels. The total number of characters defines the page length, the characters themselves represent the channels that can be slewed to. The first character must be a 1 (channel 1). See **'EL' End VFU Load, p.604**, **'Sn' Slew to Channel, p.637**, and **'VT' Slew to S6, Vertical Tab, p.645**.

'SN' Mnemonic

Native Screen Mode

Behaviour

Description *Default.* When you use **'SN'**, the "clear" mnemonics fill the cleared regions with the current colour and Reset Mode (the **'RM' Mnemonic, p.636**) resets the output to the default colours. See also: **'SX' Set Extended Screen Mode, p.641**.

'SP' Mnemonic

Standard Print

Graphical Display/Printer or Character Display/Printer

Description Use **'SP'** to switch to standard print from **'CP' Condense Print for Screen, p.597**. In ProvideX, **'CP'** or **'SP'** affect only the data that follows the mnemonic.

On GUI devices, the **'SP'** mnemonic will reset to standard printing size.

Example

```
10 open(1) "LP"
20 print (1) "NORMAL"+ ' cp ' + "COMPRESSED" + ' SP ' + "NORMAL"
30 close(1)
```

If the example above were run in ProvideX, the word COMPRESSED would be in condensed print and the word NORMAL would be in standard print. (In some other Business Basics, the font for the complete line is affected and for this example, all text would be in standard print.)

'SR' Mnemonic*Scroll Reset***GUI Display or Character Display**

Description See **'SCROLL'** *Manipulate Scroll Region*, p.638.

'SWAP' or **'WS'** Mnemonic*Swap Windows on Stack***GUI Display or Character Display**

Description Use **'SWAP'** or **'WS'** to swap the top two windows on the window stack.

'SX' Mnemonic*Set Extended Screen Mode***Behaviour**

Description Use **'SX'** to set extended screen mode. The "clear" mnemonics fill the cleared regions with the default colour and Reset Mode (the **'RM'** Mnemonic, p.636), turns off all visible attributes. See also **'SN'** *Native Screen Mode*, p.640.



Note: This feature is only included for compatibility with other languages. Refer to the *ProvideX User's Guide* for additional information on conversion and compatibility modes.

'+T' & **'-T'** Mnemonics*Text Display On/Off***GUI Display or Behaviour**

Format *Enable Text Display:* '+T'
Disable Text Display: '-T'

Description Use '+T' to re-enable the display of the text screen and have ProvideX redraw the text plane. Note that this can cause flickering in some cases. You can use '-T' to disable the display and avoid flickering. Refer to the example for the **'DIALOGUE'** Mnemonic, p.601.

'TEXT' Mnemonic

Draw Text

Graphical Display/Printer

Format `'TEXT'(x,y[,x,y],text$,attrib$)`

Where:

`attrib$` Optional attribute string. Valid codes include:

& Underscore the character following the '&' (as in hot keys)

C Centre text

F Show focus lines around text

N Numeric data alignment

R Right Justify

S Applies background colour to area directly behind text.

W Word wrap

Same as **N**

`text$` String expression.

`x,y,x,y` Point coordinates for top left and (optionally) bottom right, in graphical units. Numeric expression.

Description Use **'TEXT'** to draw (print) text in graphics mode, starting at the point set by the first `x,y` coordinates. Use graphical units or `@X(col)` and `@Y(line)` functions for the various coordinates. The **'TEXT'** mnemonic uses current **'FONT'** and colour attributes (i.e., **'RED'**, **'BLUE'**, ...).

Use the optional *second* set of `x,y` parameters to define the bottom right corner of a rectangular region for displaying the text. You can use the functions `TXH()`, *p.544*, and `TXW()`, *p.545*, to make sure the text fits the region.



Note: For Windows printers, if the current background colour is white, the output will be considered *transparent* (i.e., with no background fill).

Example

```
0010 PRINT 'FONT'("MS Serif",-11)
0020 PRINT 'GREEN', 'TEXT'(240,420,"&Hello","&")
```



'TEXTWDW' Mnemonic

Create Text Window

GUI Display or Character Display

Format `'TEXTWDW'(col,ln,wth,ht[,wdw_id],[title$][,attrib$][,OPT=val$])`

Where:

`@(col,ln,wth,ht)` Position / coordinates. Numeric expressions. Column and line coordinates for top left corner, width in number of columns and height in number of lines.

`attrib$` Attribute string. Optional. If you include attributes, use a string of one or more mnemonics to define the defaults for the window.

`title$` Optional title. String expression/literal.

`val$` Valid **OPT=** values for defining windows in a graphics environment:

c Window is a child of the current window.

S Window has Status line / Message Bar.

`wdw_id` Window's unique ID number (0 - 255).

Description Use **'TEXTWDW'** to create a text mode window under Windows. If you include a title, a box of the defined height and width will be drawn around the window and the title will be left-justified on the top line of the box.

See also **'DIALOGUE' Define / Draw Dialogue Region, p.600** and **'WINDOW' Define / Draw Window, p.647**.

Example `PRINT 'TEXTWDW' (10,10,50,10,"Title")`

'TR' Mnemonic

Terminal Read from Start

Editing

Description **'TR'** transmits an image to a program as a string. Normally, **'TR'** reads from 0,0 to the end of the screen; however, when in BBx emulation mode (**SET_PARAM 'BX'**), **'TR'** reads from 0,0 to the current cursor position.

The **'RP'** mnemonic has the same functionality as **'TR'** except that it is not affected by BBx emulation. See also **'RP' Terminal Read to End, p.636**, **'RC' Return Cursor Address, p.635**, and **'RL' Return Line Contents, p.636**.



Note: In graphics mode, if a terminal mnemonic transmits the contents of the screen to the program, the data consists of four bytes for each character. The first two bytes contain background and foreground characters, the third byte holds character attributes, and the fourth byte holds the actual character on the screen.

'TW' Mnemonic*Transmit Windows as String***Editing**

Description Use **'TW'** to transmit a list of active windows to the program as a string of 1-byte numerical values \$00\$ to \$FF\$ (to be read in the next **INPUT** statement).

'+U' & '-U' Mnemonics*Screen Refresh On/Off***Behaviour or GUI Display**

Format *Screen Refresh On: '+U'*
 Screen Refresh Off: '-U'

Description *PVX Windows only.* **'+U'** turns on screen refresh. **'-U'** turns it off. The default is *on*.



Note: Be sure to turn this back on at some point.

'UC' Mnemonic*Convert Input to Upper Case***Editing or Behaviour**

Description Use **'UC'** to convert all subsequent user input to upper case. To end upper-case conversion and allow the use of mixed case, use the **'LC'** Mnemonic, [p.617](#).

'UP' Mnemonic*Move Up One Line***Motion**

Description Use **'UP'** to move the cursor up one line.

Example `0010 PRINT "Oh_x", 'LT', "Hello", 'UP', "There"`

'+V' & '-V' Mnemonics**Control Row Highlighting****Behaviour or GUI Display**

Format *Full-line Highlight: '+V'*
 First Column Highlight: '-V'

Description The terminal mnemonic '+V' turns on full-line highlighting of a list view (report style) list box. When the user clicks anywhere on a row, the entire row will be highlighted. If '-V' is used, and the user clicks anywhere on the row, only the *first* column of the row will be highlighted.

By printing either of these mnemonics, you can adjust the highlight style for your application without having to modify each occurrence of a list view (report style) list box in your application. (See [Row Highlighting, p.191.](#))

'VT' Mnemonic**Slew to S6, Vertical Tab****Character Printer**

Description Slew to S6, Vertical Tab 'VT'. See also ['EL' End VFU Load, p.604](#), ['SL' Start VFU Load, p.640](#), and ['Sn' Slew to Channel, p.637](#).

'!W' Mnemonic**For Internal Use Only****Behaviour**

Description For Sage Software Canada use only - Included here for completeness only.

'+W' & '-W' Mnemonics**Windows-Style Windows****Behaviour or GUI Display**

Format *Use Windows Style: '+W'*
 Disable Windows Style: '-W'

Description Use '+W' to enable creation of Windows-style windows. Use '-W' to disable Windows style. When Windows-style is disabled, all windows are created with text characters.

'WA' Mnemonic*Define / Draw Window***GUI Display** or **Character Display**Description Same as **WINDOW'** *Define / Draw Window*, below.**'WC'** Mnemonic*Save/Copy Current Window***GUI Display** or **Character Display**Description Same as **'PUSH'** *Save/Copy Current Window*, p.634.**'WD'** Mnemonic*Drop Identified Window***GUI Display** or **Character Display**Description Same as **'DROP'** *Drop Identified Window*, p.602.**'WG'** Mnemonic*Make Window Current***GUI Display** or **Character Display**Description Same as **'GOTO'** *Make Window Current*, p.612.**'WHITE' & '_WHITE'** Mnemonics*Color Text***Graphical Display/Printer** or **Character Display**Format *Foreground: 'WHITE'*
Background: '_WHITE'

Description All input or output following this mnemonic will be in white foreground or background.

Example `INPUT 'WHITE', '_GREEN', "Please enter your name: ", Name$,`

'WINDOW' or 'WA' Mnemonic Define / Draw Window

GUI Display or Character Display

Format `'WINDOW'(col,ln,wth,ht[,wdw_id],[title$][,attrib$][,OPT=string$])`

Where:

`@(col,ln,wth,ht)` Position / coordinates. Numeric expressions. Column and line coordinates for top left corner, width in number of columns, height in number of lines.

`attrib$` Attribute string. Optional. If you include attributes, use a string of one or more mnemonics to define the defaults for the window.

`title$` Optional title. String expression/literal.

`string$` Optional attributes for defining windows in a graphics environment (some options are ignored in *text mode* '['+W' & '-W'](#)', [p.645](#)):

- Window has a minimize button.
- c** New window is a child of the window that launched it.
- C** Disables close button on title bar of window and eliminates the system control menu from the title bar.
- m** Enables "maximize" box in top right corner of window (only for dialogue windows created with `OPT=" * "`).
- S** Window has Status line / Message Bar.
- x** Disables close button on title bar of window and eliminates the system control menu from the title bar.
- X** Enables close button on title bar of window and supports the system control menu on the title bar.
- Z** Creates a resizable window.

`wdw_id` Window's unique ID number (0 - 255).

Description Use either '**WINDOW**' or '**WA**' in the format to draw (print) a new window. If you include a title, a box of the defined height and width is drawn around the window. The title will be left-justified on the top line of the box unless the '**AH**' system parameter is set.

ProvideX uses the `WS_BORDER` and `WS_THICKFRAME` frame styles from the Windows API with the '**WINDOW**' mnemonic. For more information, see [Windows API Frame Styles, p.579](#). See also: '[DIALOGUE](#)' [Define / Draw Dialogue Region, p.600](#) and '[TEXTWDW](#)' [Create Text Window, p.643](#)

Example `PRINT 'WINDOW'(5,5,100,40,"Title",OPT="-mSZ")`

'WM' Mnemonic*Relocate Current Window***GUI Display or Character Display**

Description Same as **'MOVE'** [Relocate Current Window, p.623](#).

'WP' Mnemonic*Wide Printer (DOS)***Character Printer**

Description *Included for completeness only.* Use **'WP'** for 40-column mode, wide printer (legacy DOS systems). See also: **'EP'** [Start Expanded Print, p.605](#).

'WR' Mnemonic*Remove Current Window***GUI Display or Character Display**

Description Same as **'POP'** [Remove Current Window, p.633](#).

'WRAP' Mnemonic*WrapAround On/Off***Behaviour or Editing**

Format *Long or short forms:*

1. Set WrapAround On: **'WRAP'("ON")** or **'BW'**
2. Set WrapAround Off: **'WRAP'("OFF")** or **'EW'**

Description Set **'WRAP' ON / OFF** to control automatic word-wrapping at the end of the line, or use **'BW'** [Begin WrapAround, p.593](#), and **'EW'** [End Wrap Around, p.606](#).

ON (Default): Information is entered in the last column of the screen or window region, the cursor will advance to the first position of the next line.

OFF: Continued output over-types at the last position on the line.

'WS' Mnemonic*Swap Windows On Stack***GUI Display or Character Display**

Description Same as **'SWAP'** [Swap Windows on Stack, p.641](#).

'WX' Mnemonic

Windows Definition Sequence

Definition

Format `'WX'=esc_seq$`

Description Use **'WX'** to define the escape sequence for creating a window on the specific terminal type. However, very few terminals have the ability to create independent windows in their display area.

For additional information regarding the use of special mnemonics (e.g., '@@', '*C', '*I', '*O', '*R', '*X', 'AT', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

'*X' Mnemonic

Program to Call on CLOSE

Definition

Format `MNEMONIC (chan)*X="prog_name[;entry][;params]"`

Where:

chan Channel or logical file number for which the mnemonic is defined.

entry Name of starting line label to use as entry point in the program. Optional. If included, add to the *prog_name* string expression (e.g., `RUN "PROG; STARTING_LABEL"`).

params Parameters (if any) needed in your **CALL**ed program.

prog_name Name of the program. String expression.

Description **'*X'** (*star-x*) contains the pathname of a program to **CALL** on the closing of a channel. When a file is closed, ProvideX issues a **CALL** to the program/entry point specified by the contents of **'*X'**. This takes place just prior to the file being actually being closed, allowing the program to alter the contents of the file if desired.

To retrieve the parameter list, the called program should reference `PGM(-3)`, which contains the complete string contained in the **'*X'** mnemonic definition.


For additional information regarding the use of special mnemonics (i.e., '@@', '*C', '*I', '*O', '*R', '*X', 'GD', 'WX') when creating a device driver, see [Dynamic Information in Mnemonics, p.580](#) or *Device Drivers* in the *ProvideX User's Guide*.

Example

```
-:LOAD "TSTX"
-:LIST
0010 LBL:
0020 MSGBOX "HELLO--"+SEP+"FIL="+STR(LFA)+SEP+"MNM="+MNM(' *X', LFA)
0030 END
-:OPEN (1) "\JUNK.TXT"
-:MNEMONIC (1) '*X'="TSTX;LBL;My backspace key is orange"
-:CLOSE (1)
```

'+X' & '-X' Mnemonics**Windows 'X' Close Button****GUI Display**

Format *Windows 'X' Close Button On: '+X'*
 Windows 'X' Close Button Off: '-X'

Description When the '+X' mnemonic is set, use of an  icon is set as the default for when new windows are created.

'XP' Mnemonic**Line Mode (DOS)****Character Display**

Description *Included for completeness only. Use 'XP' to control EGA / VGA 43/50 line mode display (legacy DOS systems). See also: 'CP' Condense Print for Screen, p.597 and 'SP' Standard Print, p.640.*

'YELLOW' & '_YELLOW' Mnemonics**Colour Text****Graphical Display/Printer or Character Display**

Format *Foreground: 'YELLOW'*
 Background: '_YELLOW'

Description All input or output following this mnemonic will be in yellow foreground or background.

Example `INPUT 'YELLOW', '_GREEN', "Please enter your name: ", Name$`

'+Z' & '-Z' Mnemonics**Text Mode Like Windows****GUI Display**

Format *Text Mode with Windows Look: '+Z'*
 Windows Look Off: '-Z'

Description The terminal mnemonic '+Z' gives text mode screens a Windows look and feel. This does not turn text mode screen into Windows screens, but it does provide a more GUI-like appearance.

'ZX' Mnemonic

Return Attributes as per BBx

Behaviour

Description

Use 'ZX' to resolve inconsistency in the attributes format and the FIN values when you run ProvideX with code designed for BBx. (In prior versions, the attributes as returned when issuing a Terminal Read from Start, 'TR', p.643 in graphics mode had the foreground and background values reversed and an incorrect window scroll region position.) If you print the 'ZX' mnemonic, ProvideX reports the attributes and FIN values consistent with BBx.

6

System Parameters

'1U'	'DB'='	'FP'	'LM'	'OM'	'QT'	'TN'	'XI'
'3D'	'DC'	'FS'='	'LP'	'OP'	'RI'	'TT'='	'XL'
'AD'	'DD'	'FT'	'LS'='	'OR'	'RN'='	'TU'	'XS'='
'AH'	'DF'='	'FU'	'LU'	'OW'='	'RP'	'TX'	'XT'
'AI'='	'DL'='	'FX'	'LW'	'PC'='	'RR'	'UL'	'ZP'
'AP'	'DP'='	'HC'	'LZ'	'PD'='	'RS'	'UM'	'!9'
'AW'	'DT'='	'HP'	'MB'='	'PE'	'SB'	'VC'	'!B'='
'B0'	'DW'='	'!O'	'MC'	'PF'	'SC'	'VM'	'!D'
'BF'='	'EG'	'!Z'	'MD'='	'PL'='	'SD'	'VP'='	'!F'
'BL'='	'EL'='	'!C'	'MP'	'PO'	'SF'	'VR'='	'!I'
'BT'	'EO'	'!M'	'MS'='	'PP'	'SK'	'VW'='	'!K'
'BX'	'ES'	'!R'	'MX'	'PQ'	'SL'='	'WB'	'!Q'='
'BY'='	'EX'	'!S'='	'NE'	'PS'='	'SP'	'WD'='	'!R'='
'CD'	'E,'	'!W'	'NI'	'PT'	'SR'	'WF'	'!S'
'CE'	'F4'	'!Z'='	'NK'	'PU'	'SS'	'WH'='	'!T'
'CF'	'FB'='	'!J'	'NL'	'PW'='	'SV'='	'WI'='	'!U'='
'CH'='	'FC'	'KF'='	'NN'	'PZ'	'SW'='	'WK'	'!V'
'CI'	'FE'	'KR'	'NR'	'Q_ '='	'SZ'='	'WL'	'!W'
'CO'='	'FF'='	'LB'='	'NS'	'Q ^ '='	'TA'='	'WP'	'!X'
'CS'	'FI'	'LC'	'NX'	'QD'='	'TB'	'WT'='	'*K'
'CT'='	'FL'	'LD'	'OC'	'QF'='	'TC'='	'WZ'='	'*L'
'CU'='	'FN'	'LE'	'OF'='	'QK'	'TH'='	'XC'	
'D0'	'FO'='	'LF'	'OL'='	'QS'	'TL'	'XF'	

Overview

System parameters are used at start-up to define a system's operation under ProvideX. Each consists of a two-character code enclosed in single quotes. Most parameters are Boolean switches (0 or negative sign indicates *off*, 1 or no sign indicates *on*), but some require specific values in order to be set. See [Setting / Resetting Parameters](#) below.



Note: In this reference, some parameters are *always* described with an equal sign to indicate that they are set (*on*) using non-Boolean values; e.g., `'BY'='` or `'DW'='`.

Most often, system parameters are set only once at the beginning of an application, typically in a start up program; however, any system parameter can be set or reset in your software at any time, whenever required.

For related information and examples, refer to the [SET_PARAM Directive, p.306](#), the [PRM System Variable, p.570](#), and the [PRM\(\) Function, p.504](#).

Setting / Resetting Parameters

The **SET_PARAM** directive, and the ProvideX *UCP utility, allow you to alter the current settings of system parameters. The specific method for setting/resetting each parameter is explained with each definition.

Examples:

```
SET_PARAM 'AH' Switches Alternative Heading on
SET_PARAM -'AH' Switches Alternative Heading off
SET_PARAM 'AH'=0 Switches Alternative Heading off.
SET_PARAM 'BY'=0 Sets the Base Year to Julian calendar base
SET_PARAM 'BY'=1970 Sets Base Year to 1970.
```

Off: Parameter shows a *negative sign* or is set to equal *0 zero*.

On: Parameter is *not negative* or is set to equal a *specific value*.

Parameter Defaults

The **PRM System Variable**, [p.570](#), can be used to return a string of the current system's parameters and their values. (Some will be hidden from the **PRM** list unless they are actually set.) **PRINT PRM** lists the following defaults in ProvideX for Windows 32-bit:

```
PRINT PRM !
-'3D',-'AD',-'AH', 'AI'=10,-'B0', 'BF'=10,-'BT',-'BX', 'BY'=1970,-'CD', 'CS',
'CT'=0, 'CU'=36,-'D0',-'DC', 'DF'=0, 'DL'=0, 'DP'=46, 'DT'=0, 'DW'=0,-'EG', 'EL'=0,
-'EO',-'ES',-'EX',-'F4', 'FB'=5,-'FC', 'FF'=0,-'FI', 'FO'=0,-'FU',-'FL', 'FP',
'FS'=138,-'FT',-'FX',-'F',-',-'I0',-'I2', 'IC',-'IM', 'IR', 'IS'=5,-'IZ',-'KR',
'LB'=4,-'LC',-'LD',-'LE', 'LS'=1,-'LU',-'LZ', 'MB'=0,-'MC', 'MF'=50,-'MP', 'NE',
-'NI',-'NK',-'NL',-'NN',-'NR',-'OC', 'OL'=25, 'OM',-'OP', 'OR', 'OW'=0, 'PC'=0,
'PD'=2,-'PO',-'PU', 'PW'=36,-'PZ', 'QF'=1, 'Q_`=2, 'Q^`=2,-'QS',-'QT',-'RI',
'RN'=1, 'RP',-'RR',-'RS',-'SC',-'SD',-'SF',-'SK', 'SL'=32,-'SP',-'SR', 'SV'=1,
'SZ'=32000,-'TB', 'TC'=0, 'TH'=44,-'TL',-'TN',-'TT',-'TU',-'TX', 'VP'=48, 'VR'=0,
'VW'=0, 'WB', 'WD'=10000,-'WF', 'WH'=0, 'WI'=1000,-'WK', 'WT'=2,-'XC',-'XF',-'XI',
-'XT',-'ZP',-'DD', '!B'=3, !U'=0,-'1U'
```

Saving / Restoring System Parameters

To avoid conflicts with other software components, we strongly recommend that you save the current settings for parameters you need to change, and then reset them when done.



Note: The **'BX'** parameter is the only exception to this recommendation, since it affects a series of system parameters. Never attempt to save/change/restore **'BX'**. It should be set/reset only at the start of a session.

Example:

```
0010 sv_ex=prm('ex'); SET_PARAM -'EX'
.....
9900 SET_PARAM 'EX'=sv_ex
```

List of System Parameters

The following parameters are listed in ASCII sort-order, except for those beginning with either an *exclamation mark* ! or an *asterisk* *, which are at the end of the chapter. Where noted, the parameters are system-specific.



Note: An *equals sign* in the heading indicates that the parameter described requires more than a simple Boolean switch to be set; e.g., '**LB**' = accepts a code number from 0 to 7 that represents a colour option. Other descriptions assume that the parameter is simply set to *on* or *off*; e.g., '**AH**' or - '**AH**' or '**AH**' = 0. See [Setting / Resetting Parameters, p.654](#).

'1U' System Parameter *Force Dedicated User Slot*

Description Forces ProvideX to use a dedicated user slot for the session. This removes the shared nature of ProvideX under Windows or WindX and also affects FacetTerm sharing.

Default *Off.* ProvideX attempts to use shared user slots for the session. That is, the same terminal is considered a single user regardless of the number of invocations. **TCB(27)** returns the User Slot Number this session is using and determines whether it is dedicated, shared, or a background task. Refer to the [TCB\(\) Function, p.534](#).



Note: Once you set '**1U**' *on*, you can't turn it *off* unless the application terminates.

'3D' System Parameter *3D in Windows*

Description Displays 3D look in Windows. Use the '**2D**' Mnemonic, [p.585](#), '**3D**' Mnemonic, [p.586](#), or the '**4D**' Mnemonic, [p.586](#) instead.

Default *Off.*

'AD' System Parameter *Auto-DIM Array*

Description Auto-DIM's an array.

Default *Off.* You must use the [DIM Directive, p.86](#).

Example

```
SET_PARAM 'AD'
LET X$[ALL]=Y$[ALL]
```

'AH' System Parameter *Alternative 'WINDOW'/'BOX' Heading*

Description	Uses alternative 'WINDOW'/'BOX' heading (with the title in inverse video, centred on the top line).
Default	<i>Off.</i> The title is text, left-justified on the top line of the 'WINDOW' / 'BOX'.

'AI'= System Parameter *Automatic Line-Number Increment*

Description	Assigns a value for the default line numbering increment for LOAD , RUN , CALL , or PERFORM directives with an ASCII text program and/or when you use the AUTO directive.
Default	'AI'=10 (i.e., auto increment by 10).



Note: 'AI' is reset to the default increment (10) when a **START** is issued. If the 'UL' parameter is set, it will override the 'AI' parameter for line numbering .

'AP' System Parameter *Auto-Enable PDF Output*

Description	Processes options selected during *WINPRT* Printer Selection dialogue to be used for *PDF* output instead.
Default	<i>Off.</i>

'AW' System Parameter *Alternate WINPRT_SETUP*

Description	<i>PVX Windows/WindX Only.</i> Switches WINPRT_SETUP from implementing legacy 16-bit compatible functions to the Windows-recommended API calls (which observe account privileges). For more information, see WINPRT_SETUP Directive, p.376 .
Default	<i>Off.</i>

'BO' System Parameter *Base Zero for Level / Window*

Description	Sets zero as the base number for level and window. For WindX environments, this parameter must be set on both the client and the server.
-------------	--

Default *Off.* The base number for level and window is 1.



Note: This setting affects the **HWN()** function, which relies on the base window number. (See [HWN\(\) Function, p.455.](#))



Warning: If the application is run with '**B0**' set, ensure that it is set on both the UNIX host and the WindX PC. Otherwise, the wrong windows will be addressed. Set '**B0**' either by using **-B0** as an argument on the WindX PC's startup command line or as a statement from the host; i.e., `EXECUTE "[wdx]SET_PARAM 'B0' "`.

'BF'= System Parameter

Common File Buffers

Description Assigns the number of common database Keyed I/O buffers to be maintained by the system.

Default '**BF**'=10.



Note: When '**BF**'=0 (zero, no common buffers), ProvideX uses the setting in '**FB**' [Dedicated File Buffers, p.665.](#)

'BL'= System Parameter

Break Lines in Listings

Description Specifies breaks in long lines at the first comma, plus sign, or the words **OR** and **AND**, which occur on or after the number of columns specified.

Default '**BL**'=0.

'BT' System Parameter

Binary Test: 1st Read

Description Sets binary test for serial files on first read. If a size is indicated (**ISZ=option**), then the file is treated as binary. Otherwise, the file is treated as record oriented.

Default *Off.*



'BX' System Parameter

BBx Emulation

Description Sets ProvideX to BBx emulation mode. This automatically resets certain system parameters to behaviour required to run programs designed for BBx.



Note: Use the **'LZ' Suppress Leading Zeros**, [p.673](#), to control rounding of leading zeros. (This functionality was previously controlled by the **'BX'** parameter.)

The following parameters are affected by the **'BX'** parameter: **'AD'**, **'BO'**, **'BT'**, **'BY'=**, **'CD'**, **'DC'**, **'EX'**, **'FF'=**, **'IO'**, **'JC'**, **'KR'**, **'MP'**, **'NR'**, **'OP'**, **'QS'**, **'RS'**, **'WK'**, **'XF'**. For WindX environments, this parameter must be set on both the client and the server.

Default *Off.* ProvideX is in standard mode (**'FF'=0**).

'BY'= System Parameter

Base year

Description Defines the base year for the **JUL() Function**, [p.463](#), and the **DTE() Function**, [p.422](#).

Default **'BY'=1970**.



Note: The **JUL()** function uses January 1 of the base year as day 0 zero.

Also: If you **SET_PARAM 'BY'=0 zero**, the **JUL()** and **DTE()** functions operate under the Julian calendar where day 0 is around **4713 BC**.

'CD' System Parameter

Check Current Directory

Description Checks the current directory first before checking the prefix list for a file.

Default *Off.* Searches current directory after the prefixes (normal method).

'CE' System Parameter

Obsolete

Description *Deprecated.* Use the **'NE' System Parameter**, [p.676](#).

'CF' System Parameter

Bypass Console Flush

Setting *PVX UNIX/Linux Only.* Bypasses console flush for print statements. Use **'CF'** to improve display performance (i.e., no waiting for OS acknowledgment after every **PRINT** statement).

Default *Off.* Performs a console flush on print statements.

'CH' = System Parameter*Hover Colour*

Description Controls the text colour of an object that has a hover attribute when the mouse is *not* over the object. Supported options include:

0 - Current	8 - Dark Gray
1 - Light Red	9 - Dark Red
2 - Light Green	10 - Dark green
3 - Light Yellow	11 - Dark Yellow
4 - Light Blue	12 - Dark Blue
5 - Light Magenta	13 - Dark Magenta
6 - Light Cyan	14 - Dark Cyan
7 - White	15 - Gray

Default 'CH'=12 (Navy blue).

'CI' System Parameter*Cache IOList*

Setting Enhances performance when reading files opened with an **IOL=*** by caching variables associated with the internal IOList.

Default *On.*

'CO' = System Parameter*Mouse Over Colour*

Description Controls the text colour of an object that has a hover attribute when the mouse is over the object.

Supported options include:

0 - Current	8 - Dark Gray
1 - Light Red	9 - Dark Red
2 - Light Green	10 - Dark green
3 - Light Yellow	11 - Dark Yellow
4 - Light Blue	12 - Dark Blue
5 - Light Magenta	13 - Dark Magenta
6 - Light Cyan	14 - Dark Cyan
7 - White	15 - Gray

Default 'CO'=4 (Blue).

'CS' System Parameter

Coloured Syntax

Setting	<i>On.</i> Displays program with coloured syntax when using the LIST Directive, p.176 , and the LST() Function, p.477 . <i>Off.</i> Does not display coloured syntax.
Default	<i>On.</i>
See Also	'*H' Mnemonic, p.613 .

'CT'= System Parameter

Character Time-out

Description	<i>PVX Windows/WindX Only.</i> Adjusts character time-outs for slower LPT devices. This value specifies the multiplier, in milliseconds, used to calculate the total time-out period for write operations to LPT devices. For each write operation, this value is multiplied by the number of bytes to be written.
Default	'CT'=0

'CU'= System Parameter

Currency Symbol

Description	Assigns the currency symbol / character to be used in formatted numeric data I/O. Use the ASCII value of the character.
Default	'CU'=36 or 'CU'=ASC("\$")

'D0' System Parameter

Divide by Zero

Description	Sets ProvideX to return 0 zero for divisions by zero.
Default	<i>Off.</i> Division by zero results in Error #40: Divide check or numeric overflow.

'DB'= System Parameter

Dynamic File Buffers

Description	Controls various aspects of the dynamic buffering logic. Valid settings are as follows: <ul style="list-style-type: none"> 1 - Share O/S file handles when files are opened on multiple channels. 2 - Use local buffers when files are shared. 4 - Allocate optimum number of buffers when updating files. 8 - Add common buffers when single update requires more buffers than are available. 16 - Maintain key buffers and inventory pages by dropping data buffers first.
-------------	---

The above values are additive; e.g., to set **1** and **4**, use `SET_PARAM 'DB'=5`.

Default **'DB'=31**. All settings (**1+2+4+8+16**).

'DC' System Parameter

Destructive Cursor

Description Sets destructive cursor. (Moving from left to right, replaces intervening characters with spaces up to the new cursor position).

Default *Off*. The cursor jumps over the intervening characters.

Example

```
SET_PARAM 'DC'=0
PRINT @(15), "B", @(10), "A", @(20), "C"
```

With **'DC' off**, the screen output is " A B C". The cursor jumps first to column 15 and prints the B, jumps to column 10 and prints the A, then finally jumps to column 20 and prints the C.

```
SET_PARAM 'DC'
PRINT @(15), "B", @(10), "A", @(20), "C"
```

With **'DC' on**, ProvideX uses a destructive cursor (line print mode, as in BBx and some MAI systems). The preceding example would result in " A C", because the output driver simply issues spaces when advancing on the same line. With **'DC' on**, after printing the B and A, the driver uses 9 spaces to position itself at column 20 to print the C. The destructive cursor overwrites the B with the space.

'DD' System Parameter

Convert Directory Delimiter

Description Replaces reverse slash (\) in Windows path names with a standard slash (/).

Default *Off*. No replacement of delimiter occurs.

'DF'= System Parameter

Enforced Delay Time after 'FF'

Description Adds automatic delay times after a each **'FF'** without having to insert **WAIT** statements into your application. Use this parameter to have ProvideX return control to the OS for longer periods of time. **'DF'=num** is a numeric value from 0 to 1000:

0 indicates no delay (default)

1 indicates a forced **WAIT 0** when the event occurs.

All other values from 2 through 1000, indicate the *number of 100ths of a second* to delay on each event occurrence. (See also: **'FF' Mnemonic, p.607.**)

Default 'DF'=0. *No Delay.*



Note: The 'DF' delay only applies to 'FF' when sent to a device, Windows print spooler, or a WindX-connected file.

'DL'= System Parameter *Enforced Delay Time after 'LF'*

Description Adds automatic delay times after a each 'LF' without having to insert **WAIT** statements into your application. Use this parameter to have ProvideX return control to the OS for longer periods of time. 'DL'=num is a numeric value from 0 to 1000:

0 - indicates no delay (default)

1 - indicates a forced **WAIT 0** when the event occurs.

All other values from 2 through 1000, indicate the *number of 100ths of a second* to delay on each event occurrence.

Default 'DL'=0. *No Delay.*



Note: The 'DL' delay only applies to 'LF' when sent to a device, Windows print spooler, or a WindX-connected file.

'DP'= System Parameter *Decimal Point Symbol*

Description Assigns decimal point symbol / character for use in formatted numeric data. Use the ASCII value of your character. For WindX environments, this parameter must be set on both the client and the server. In order for this format to be applied in numeric masking, the mask must be identified as numeric using the # character (see [Numeric Format Masks, p.814](#)).

Default 'DP'=46 or 'DP'=ASC("."), the decimal point.



Note: The 'DP' setting is used if I/O is formatted in **INPUT**, **OBTAIN**, **PRINT** and the **STR()** function (and ignored for unformatted I/O). It is always ignored by the **NUM()** function and when using **WRITE**, **READ**, **FIND** or **EXTRACT** directives in converting numeric data.

'DT'= System Parameter *Device Time-out*

Description Sets the number of seconds to wait for output to a device before a device error is returned.

Default 'DT'=0

'DW' = System Parameter**Delay Time after 'WI'**

Description Adds automatic delay times after 'WI' is exhausted without having to insert **WAIT** statements into your application. Use this parameter to have ProvideX return control to the OS for longer periods of time. **'DW'=num** is a numeric value from 0 to 1000:

0 - indicates no delay (default)

1 - indicates a forced **WAIT 0** when the event occurs

All other values from 2 through 1000, indicate the *number of 100ths of a second* to delay on each event occurrence.

Default **'DW'=0. No Delay**

'EG' System Parameter**End Generation of Error #29**

Description Sets to ignore invalid mnemonics instead of generating Error #29.

Default *Off.* Generates Error #29: Invalid Mnemonic or position specification for invalid mnemonics.

See Also ['BG' Mnemonic, p.589](#),
['EG' Mnemonic, p.603](#).



Note: The **'EG' parameter** affects all file channels while the **'BG'** and **'EG' mnemonics** affect specific channels.

'EL' = System Parameter**Encryption Level**

Description Returns the current encryption setting, or sets encryption to a new level for password protected programs. This allows ProvideX to load/run any password-protected programs created with encryption levels 0 through 4 (5 levels in total). See also [PASSWORD Directive, p.239](#).

Default *Varies by version of ProvideX.*

'EO' System Parameter*Embedded 'EO' Mnemonics*

Description Sets handling of embedded 'EO' mnemonics (to end output transparency). ProvideX scans strings following a 'BO' mnemonic for embedded ESC+"EO". This parameter simplifies the conversion to ProvideX from other languages.



Note: When 'EO' is *on*, any ESC+"EO" sequence embedded in the data will terminate output transparency mode.

Default *Off.* The strings are not checked for embedded 'EO' mnemonics.

Example The example below shows a typical use of the 'BO' (begin) and 'EO' (end) output transparency mnemonics. The 'EO' mnemonic is not embedded in the string, so ProvideX will automatically recognize it and terminate a 'BO' mnemonic.

```
PRINT( chan ) 'BO' , "some sequence" , 'EO'
```

In the next example, the 'EO' mnemonic is embedded in the string (appended using the +plus sign).

```
PRINT( chan ) 'BO' + "some sequence" + 'EO'
```

In this case, if the 'EO' system parameter is *off* (the default), the embedded 'EO' mnemonic is ignored. When the 'EO' parameter is *on*, ProvideX recognizes the embedded mnemonic and ends output transparency.

'ES' System Parameter*Display OS Errors in Command Mode*

Description Sets display of OS extended status / errors in Command mode. 'ES' is used primarily by developers to enforce good programming standards.

Default *Off.* No display of OS errors in Command mode.

'EX' System Parameter*Apply Execute at Level 0*

Description Sets EXECUTE directives to affect the program at level 0.

Default *Off.* ProvideX changes the current program.



Note: If the EXECUTE directive starts with a line number, ProvideX modifies the current program and the line becomes part of the current program, unless the system parameter 'EX' is *on*, in which case it modifies the program at level 1.

See Also EXECUTE Directive, *p.123*.

'F,' System Parameter *Suppress Commas on Numeric Overflow*

Description Suppresses commas when numeric formats overflow (i.e., thousands separators are stripped) and retry the format. For more information, see [Data Format Masks, p.813](#).

Default *Off.* Generates an Error #43: Format mask invalid on format mask overflows. (The error is not returned in cases like 0100 INPUT EDIT "Enter value: ", INV_AMT: "\$###, ##0.00" where the directive allows for more input than the mask accommodates.)

'F4' System Parameter *Return CTL=4 for Exit*

Description Returns CTL=4 when the user chooses the Windows close button in a window.

Default *Off.* Returns CTL=-1999.

'FB'= System Parameter *Dedicated File Buffers*

Description Assigns the number of dedicated file buffers to use for each file.

Default 'FB'=5



Note: ProvideX uses the 'FB' setting if 'BF'=0 is set (zero, no common buffers).

'FC' System Parameter *Force File Commit*

Description *DOS Only.* Forces OS file-commit to any updated Keyed file on an INPUT or WAIT statement.

Default *Off.* No forced flush.

'FE' System Parameter *Obsolete*

Description *Deprecated.* Use the 'FI' System Parameter, p.666.

'FF' = System Parameter**File Format**

Description Defines the format of information returned by the **FID()** function. There are 5 possible formats:

- 0** - ProvideX standard format
- 1** - Thoroughbred emulation
- 2** - Rexon emulation
- 3** - BBx emulation.
- 4** - BBx emulation. Sequential files are reported as binary files for the **FID()** function, \$03\$ rather than a \$01\$ in position (1,1) of the FID.

Default **'FF'=0** (zero, for ProvideX standard format)



Note: When **'FF'** is set to 0 or 4 and the **'PO'** system parameter is switched *on*, the **FID()** and **FIB()** functions return the original path used when the file was opened.

Also: The **FID()** and **FIN()** format layouts will be changed whenever there is a change to the **'FF'** system parameter.

See Also **'PO'** System Parameter, *p.680*, **FIB()** Function, *p.434*, **FID()** Function, *p.438*, and **FIN()** Function, *p.441*.

'FI' System Parameter**Ignore Format Mask Error**

Description Suppresses **Error #43** on format mask overflows. This parameter is not intended to control errors generated when the mask character itself is invalid. For more information, see **Data Format Masks**, *p.813*.

Default *Off.* Generates an **Error #43: Format mask invalid** for format mask overflow errors.

'FL' System Parameter**Filename in Lower Case**

Description Passes filenames to the OS in lower case.

Default *Off.* Filenames are passed in original case.



Note: When the **'FL'** system parameter is set to *on*, **'FU'** is turned off automatically.

Also: The **'FN'** system parameter may override the **'FL'** setting.

See Also **'FN'** System Parameter, *p.667*, **'FU'** System Parameter, *p.668*, and the **FFN()** Function, *p.432* (for a UNIX example using these parameters for case-insensitive searches).

'FN' System Parameter *Filename As-Is: No Case Conversion*

Description	<i>On.</i> Passes filenames to the OS with no case conversion. <i>Off.</i> Case conversion is governed by the current setting of either 'FL' (Forces Lower Case) or 'FU' (Forces Upper Case).
Default	<i>On</i> , unless 'FL' or 'FU' are <i>on</i> . The 'FN' setting is not displayed in the list returned using PRINT PRM .
See Also	'FL' System Parameter, p.666 , 'FU' System Parameter, p.668 , and the FFN() Function , p.432 (for a UNIX example using these parameters for case-insensitive searches).

'FO'= System Parameter *Format Overflow Character*

Description	Assigns the format overflow symbol / character you want returned as fill on format errors. Use the ASCII value of your character. For example, to assign the <i>asterisk</i> * as the overflow character, apply either 'FO'=42 or 'FO'=ASC("*"). For more information, see Data Format Masks, p.813 .
Default	'FO'=0 <i>zero</i> . ProvideX returns <code>Error #43: Format mask invalid</code> .
Example	In the example below, ProvideX displays 6 asterisks instead of causing an error: <pre>SET_PARAM 'FO'=ASC("*") PRINT 123456.78: "##0.00" *****</pre>

'FP' System Parameter *Floating Point*

Description	<i>On.</i> Uses floating point hardware automatically if it's available on your computer. <i>Off.</i> ProvideX leaves 'FP' switched <i>off</i> if floating point hardware is not available.
Default	<i>On</i> if your computer has a floating point processor. Otherwise it is <i>off</i> .

'FS'= System Parameter *Default Field Separator*

Description	Assigns the default field separator value for the SEP system variable. Use the ASCII value of your character, either as a decimal value or by using the ASC() Function , p.396 . You can also use the ASCII value of the character that is currently in the DLM System Variable , p.558 .
Default	'FS'=138 (\$8A\$, not a printable character) or 'FS'=ASC(SEP).

Example In this example, the current value in the **DLM** system variable (the backslash, "\") replaces the **SEP** value \$8A\$ (i.e., **CHR(138)**) as the default field separator:

```
- :a$=sep;?hta(a$),asc(sep)
8A 138
- :set_param 'fs'=asc(dlm);? hta(sep),asc(sep)
5C 92
```

'FT' System Parameter

Trapping the F10 Key

Description To use the **F10** key to activate the menubar (the standard Windows default). In effect, **F10** acts as an **Alt** key.

Default *Off.* **F10** behaves like any other function key. (ProvideX overrides the Windows default.)

'FU' System Parameter

Filename in Upper Case

Description *On.* Passes filenames to the OS in upper case.

Default *Off.* Filenames are passed in original case.



Note: When the 'FU' system parameter is set to *on*, 'FL' is turned off automatically.
Also: The 'FN' system parameter may override the 'FU' setting.

See Also ['FL' System Parameter, p.666](#), ['FN' System Parameter, p.667](#), and the [FFN\(\) Function, p.432](#) (UNIX example using these parameters for case-insensitive searches).

'FX' System Parameter

Force EXTRACT

Description Returns Error #13: File access mode invalid if the program does not **EXTRACT** before all rewrites.

Default *Off.* Allows writes to a file with an extract of the record.

'HC' System Parameter

Obsolete

Description *Deprecated.* Use the ['SC' System Parameter, p.686](#).

'HP' System Parameter*LibHaru *PDF**

Description	Switches between the original and LibHaru implementations of <i>*PDF*</i> , <i>p.744</i> . LibHaru is an open-source library that enables expanded font support when generating PDFs. This parameter is WindX aware.
Default	<i>On</i> . LibHaru enabled.

'IO' System Parameter*Ignore Null Substring (No Error 47)*

Description	Treats all substrings of 0 <i>zero</i> length, starting at offset 0 <i>zero</i> , as valid NULL strings and not generate Error #47.
Default	<i>Off</i> . A null substring (e.g., X\$(0,0)) results in an Error #47: Substring reference out of string.

'I2' System Parameter*Ignore Max. Record Count (No Error 2)*

Description	Ignores maximum record counts. No Error #2 is reported when adding records to keyed or indexed files.
Default	<i>Off</i> . To report Error #2: END-OF-FILE on read or File full on write when the maximum record count is exceeded.

'IC' System Parameter*Ignore Case*

Description	Sets the command line scan function *[] to scan for matches regardless of upper/ lower case.
Default	<i>On</i> . The scan function *[] ignores case sensitivity for searches.

'IM' System Parameter*Insert Mode for Input*

Description	Remembers the current insert mode for the next input. This affects the console command line only.
Default	<i>Off</i> . All text mode inputs will start in overstrike mode.

'IR' System Parameter *Insert Mode Reset (Decimal Point)*

Description Reset insert mode upon entry of a decimal with input to the right of the decimal during formatted numeric entry.

Default *On.*

'IS'=' System Parameter *CTL for Input Ending on SIZ='*

Description Assigns the control value (CTL) to be returned when input terminates on a **SIZ='** clause.

Default 'IS'=5

'IW' System Parameter *Terminate Invoke Wait*

Description Controls whether switching focus back to a ProvideX task will terminate an **INVOKE WAIT**. When **'IW'** is enabled, ProvideX will wait until the task launched by the **INVOKE WAIT** is completed. The **'IW'** parameter is WindX aware .

Default *On.*

'IZ'=' System Parameter *Ignore Max. Memory Setting*

Description To ignore the maximum memory setting defined by either the **START Directive**, [p.328](#), or the **'SZ'=' System Parameter**, [p.688](#).

Use the **'IZ'** parameter when converting an application from another language where the values on the **START** directives may be incompatible with ProvideX standards.

Default *Off.* Memory use will be limited via **START** and **'SZ'='** settings.

'JC' System Parameter *Obsolete*

Description *Deprecated.* Use the **'DC'** System Parameter, [p.661](#).

'KF' = System Parameter**Keyed File Format**

Description	To set the default format type for creation by the KEYED directive, where: 'KF'=0 , the KEYED directive always creates VLR or FLR-based files. 'KF'=1 , the KEYED directive creates EFF formatted files <i>with</i> a 2GB limit. 'KF'=2 , the KEYED directive creates EFF formatted files <i>without</i> the 2GB limit on platforms that provide Large File Support (LFS), 64-bit addressing. Setting this parameter on a system that does not provide LFS will automatically change to a 'KF' value of 1.
Default	'KF'=0.

'KR' System Parameter**Keyed File I/O Emulates BBx**

Description	To switch the Keyed file I/O module to BBx emulation mode. The KEP() Function , p.469 , returns the key of the record prior to the next record to be read. KEY() functions do not switch keys.
Default	<i>Off.</i> See normal behaviour for the KEYED Directive , p.166 .

'LB' = System Parameter**Colour for Line # in Break Points**

Description	Sets numeric colour code. Valid colour options for line number in break points: <table> <tr> <td>0 - Black</td> <td>4 - Blue</td> </tr> <tr> <td>1 - Red</td> <td>5 - Magenta</td> </tr> <tr> <td>2 - Green</td> <td>6 - Cyan</td> </tr> <tr> <td>3 - Yellow</td> <td>7 - White</td> </tr> </table>	0 - Black	4 - Blue	1 - Red	5 - Magenta	2 - Green	6 - Cyan	3 - Yellow	7 - White
0 - Black	4 - Blue								
1 - Red	5 - Magenta								
2 - Green	6 - Cyan								
3 - Yellow	7 - White								
Default	'LB'=4 , Blue.								

'LC' System Parameter**List Variables in Lower Case**

Description	Sets the LST() Function , p.477 , and the LIST Directive , p.176 , to return variable names in lower case.
Default	<i>Off.</i> Variable names are listed in upper case.
See Also	'MC' Mixed Case , p.674 .

'LD' System Parameter *List Directives in Lower Case*

Description Sets the **LIST()** Function, [p.477](#), and the **LIST Directive**, [p.176](#), to return directive names in lower case.

Default *Off.* Directives are listed in upper case.

'LE' System Parameter *SAVE / LIST Indent Statements*

Description Indents program statements for saves to a serial file (**SAVE Directive**, [p.295](#)) and for the **LIST Directive**, [p.176](#).

Default *Off.*

'LF' System Parameter *Long Form Variables*

Description Sets the compiler to allow long variable names. The directive **LONG_FORM**, [p.201](#) works the same as **SET_PARAM 'LF'**. Use either **SET_PARAM -'LF'** or the **SHORT_FORM Directive**, [p.325](#) to cancel.



Note: System parameters **'LF'** and **'SF'** have reciprocal values and are directly related. Any change to **'SF'** results in an opposite change to **'LF'** and vice-versa.

Default *On.* The compiler allows long variable names

'LM' System Parameter *List, Show Matched Strings*

Description Highlights matches to a currently-defined search string in a **LIST** operation to the console (**LIST Directive**, [p.176](#)). For further information on the search utility, refer to **Punctuation/Syntax**, [p.25](#).

Default *Off.*

'LP' System Parameter *Obsolete*

Description *Deprecated.* Use the **'SP'** System Parameter, [p.687](#).

'LS' System Parameter*Colour for Line with Syntax Error*

Description Sets numeric colour code used for syntax errors. Valid colour options for program line number with syntax errors:

0 - Black	4 - Blue
1 - Red	5 - Magenta
2 - Green	6 - Cyan
3 - Yellow	7 - White

Default 'LS'=1, Red.

'LU' System Parameter*Lock Unnecessary: Serial Files*

Description Makes lock unnecessary on writing to serial files. (No Error #13 on WRITE without lock.)

Default *Off.* A "sticky" parameter. ProvideX generates Error #13: File access mode invalid if there is no lock on the first write to an open file number.



Note: File access mode is checked once, on the first write after opening. An error won't be generated on a subsequent unlocked write if the file number is continuously open between writes (even if the parameter is switched off between writes).

'LW' System Parameter*For Internal Use Only*

Reserved For Sage Software Canada use only - attribute included here for completeness only.

'LZ' System Parameter*Suppress Leading Zeros*

Description Suppresses leading zeros for numeric values. This was previously controlled by the **'BX' System Parameter**, p.658. Printing an expression such as 1 / 4 normally yields a return value of 0 . 25. With **'LZ'** enabled, the return value would be . 25.

Default *Off.*

'MB' = *System Parameter* *MegaBytes: File Segment Size*

Description Controls segment size in multi-segmented files. **'MB'** must be set to a value to activate the multi-segmented file feature. ProvideX calculates the approximate size of each segment in megabytes, based on the block size of a file:

$$\text{Segment size in bytes} = 512 + ((bksz - 6) * bksz)$$

bksz above represents the block size of the file. You can use the **BSZ=** option when you create the file, to override the default block size.

The following table lists the maximum number of segments allowed for a file based on valid block sizes (1 to 31 kilobytes):

<i>Block Size</i>	<i>Segments</i>	<i>Block Size</i>	<i>Segments</i>	<i>Block Size</i>	<i>Segments</i>
31K	124	30K	120	29K	116
28K	112	27K	108	26K	104
25K	100	24K	96	23K	92
22K	88	21K	84	20K	80
19K	76	18K	72	17K	68
16K	64	15K	60	14K	56
13K	52	12K	48	11K	44
10K	40	9K	36	8K	32
7K	28	6K	24	5K	20
4K	16	3K	12	2K	8
1K	4				



Note: This feature is available for variable-length records (VLR) format only.

Default **'MB'=0**

'MC' *System Parameter**Maintain Case*

Description Allows mixed case for variable names and line labels in program listings; e.g.,

```
10 ThisIsFirst=10
20 THISISFIRST=20
LIST
10 ThisIsFirst=10
20 ThisIsFirst=20
```

Once **'MC'** is set, the first instance of a variable will establish the case setting for all subsequent uses. To change the case of a variable name, all references to the variable must be removed and the program must be saved and reloaded.

Default *Off.* Variable names and line labels are maintained in uppercase only.



Note: The case setting is also affected by the **'LC' System Parameter**, p.671.

'MF'= System Parameter

Multi-Line Size Factor

Description Reduces or increases the multi-line size factor. This affects the amount of white space appearing above and below multi-line text (where accents and descenders are displayed).



Note: By default, Windows creates multi-line input 50% larger than the font size to allow for white space above and below. While the 'MF' parameter allows you to adjust this default, we do not recommend it.

Default 'MF'=50

'MP' System Parameter

Returns Positive Modulus Value

Description To have the **MOD()** function always return a positive number.

Default *Off.* See normal behaviour for the **MOD() Function**, p.483.

'MS'= System Parameter

Memory for Program Swap

Description *Included for completeness only.* Defines the maximum amount of conventional memory to be used as program swap space (legacy DOS systems). If using Extended Memory, the value should be left at zero. See **'XS' Extended Memory (KB)**, p.697.

Default 'MS'=0

'MX' System Parameter

User-Defined Message Box

Description Allows use of a customizable message box (`msgbox.gui`) instead of the standard message box Windows API. For details, refer to the **MSGBOX Directive**, p.212.



Note: When 'MX' is set, **MSGBOX** commands entered in console mode or executed within an **EXECUTE** command *cannot* be followed by any other command (as **MSGBOX** will be executing a **CALL** without a return address).

Default *On* (if `*ext/msgbox.gui` exists), *Off* (if `*ext/msgbox.gui` does not exist).

'NE' System Parameter *Subprogram Error Report*

Description *On.* Errors in a subprogram are reported in the subprogram.
Off. Errors in a subprogram are returned to the ultimate parent program.

Default *Off.*

'NI' System Parameter *Ignore Blanks in Numeric Fields*

Description Ignores spaces in numeric fields.

Default *Off.* Spaces in numeric fields return an Error #26: Variable type invalid.

'NK' System Parameter *Null Key Stripping*

Description Strips trailing nulls from a key; i.e., key values \$00\$, \$0000\$ and \$000000\$, become \$\$.

Default *Off.* No stripping occurs.

'NL' System Parameter *Suppress LET Directive in Listings*

Description Suppresses the directive [LET Directive, p.173](#), in listings.

Default *Off.* LET directive is displayed in listings.

'NN' System Parameter *No Line Numbers as References*

Description Invalidates statements that reference line numbers. For instance, GOTO 1050 is not allowed. ProvideX returns Error #85: Program does not support line numbers. A line label is required; i.e., GOTO Label_1050.

Default *Off.* Statements can reference line numbers. (No Error #85).

'NR' System Parameter *No Intermediate Rounding on Division*

Description Prevents intermediate rounding on division. (This has no affect on other operations, such as multiplication.)

Default *Off.* Intermediate rounding is performed.

'NS' System Parameter*No Swapping*

- Description** *Included for completeness only.* Prevents the swapping to extended or expanded memory (legacy DOS systems).
- Default** *Off.* Swapping will occur.

'NX' System Parameter*Obsolete*

- Description** *Deprecated.* Use the **'XT'** System Parameter, p.697.

'OC' System Parameter*Commit Prior to OPEN Directive*

- Description** Forces the system to commit all file updates and locks prior to **OPEN** directive.
- Default** *Off.* Outstanding file updates will be handled when retrieved.



Note: Use of this parameter may slow performance.

'OF'= System Parameter*Maximum Size Before Output Flush*

- Description** *PVX UNIX/Linux Only.* Defines the maximum transmission size in bytes to be sent to an output device before forcing an output flush. Some UNIX systems and drivers can suffer overrun conditions if this parameter is not set.
- Default** **'OF'=255** on AIX systems.

'OL'= System Parameter*Maximum Buffers for OPEN LOAD*

- Description** Controls the default maximum number of Keyed buffers for files on **OPEN LOAD**.
- Default** **'OL'=25.**

'OM' System Parameter*Old Style Mask*

Description Uses old style **MSK()** function logic. **MSK()** is now compatible with the UNIX **grep** command.

Default *On.* See normal behaviour for the **MSK() Function**, p.486.

'OP' System Parameter*Return Original Program Name*

Description Sets the **PGM()** function to return the original program name.

Default *Off.* **PGM()** will return the full pathname of a program.

'OR' System Parameter*Full OS Path for Rename*

Description *On.* Treats the destination (second) parameter in the **RENAME** directive as containing a fully expanded OS path for the renamed file.

Off. Standard ProvideX search rules and file-naming conventions apply to the renamed file.

Default *On.* Refer to the **RENAME Directive**, p.282.

'OW'= System Parameter*Owner Application Code*

Description Assigns an owner application code to all saved programs.

Default **'OW'=0** (zero, no code set).

'PC'= System Parameter*Program Load Caching*

Description Defines the maximum number of programs to remain in program cache. For **'PC'=nnn**, ProvideX will maintain the last *nnn* programs (subprograms) in memory, thus reducing the time required to reload them.

The program cache is automatically flushed when dropping to a console prompt. Also, any changes to the **'PC'** setting will purge the cache and reload programs from disk. For example, use the following to *reset* cached programs so that changes on disk are reloaded:

```
SET_PARAM 'PC'=PRM('PC')
```

Default 'PC'=0 (no programs in cache).



Note: Since the program cache setting is local to the machine, any changes to the "disk file" will not be reflected across the network until the user initiates a new session, changes the 'PC' setting, or saves the program locally.

See Also [ADDR Directive, p.30.](#)

'PD'= System Parameter *Default Precision for Current Session*

Description Assigns the default number of decimal places for the current session (**PRECISION** range 0 to 18).

Default 'PD'=2



Note: This parameter is reset to 2 if a **START** command is issued.

'PE' System Parameter *Password Error Control*

Description Attempts to read a file when an invalid password is supplied for a data file that has **WRITE** or **WRITE AND ON DATA** (read-only) privileges. Refer to the **PASSWORD Directive, p.239** for more information.

Default *Off.* Generates an error when an invalid password is encountered.

'PF' System Parameter *EMS Page Frame*

Description *Included for completeness only.* Sets use of EMS page frame for work space, increasing the available memory by 64KB (legacy DOS systems).

Default *On,* if EMS is available on the machine.

'PL'= System Parameter *Program Libraries*

Description Limits the number of *cached* program libraries.

Default 'PL'=10.

'PO' System Parameter*Path Original*

Description Returns the original pathname specified in the [FID\(\) Function, p.438](#).

Default *Off*. The expanded pathname is returned.



Note: When 'FF' is set to 0 or 3 and 'PO' is *on*, the [FID\(\)](#) and [FIB\(\)](#) functions return the original path used when the file was opened.

See Also ['FF'= System Parameter, p.666](#), [FIB\(\) Function, p.434](#), and [FID\(\) Function, p.438](#).

'PP' System Parameter*Prompt for Password*

Description *On*. Prompts the user to enter a password when an attempt is made to open a passworded file without specifying a **KEY=** value or on a null **KEY=** value. Refer to the [PASSWORD Directive, p.239](#) for more information.

Off. No prompts for password.

Default *On*.

'PQ' System Parameter*Password Queue*

Description Sets the maximum number of files that can be recorded in the password queue. Refer to the [PASSWORD Directive, p.239](#) for more information.

The queue stores the filename and password for each passworded file that has been successfully opened. All attempts to open a passworded file without specifying a password will be verified with the queue to see if the password has been previously supplied. If the filename appears in the queue, the password will be re-applied from the existing entry.

Specifying a password value for a filename that already appears in the queue causes the entry to be removed and the password to be re-verified. To reset the queue, use: **SET_PARAM 'PQ'=PRM('PQ')**.

Default **'PQ'=100**

'PS'= System Parameter Maximum Program Size (KB)

Description	<i>Included for completeness only.</i> Defines the size (in KB) of the "program load" region (legacy DOS systems). Setting this enables the swapping logic and should be set to just larger than the largest program used by the application to a maximum of 63KB.
Default	'PS'=0

'PT' System Parameter Obsolete

Description	<i>Deprecated.</i> Use the 'QT' System Parameter, p.683 .
-------------	---

'PU' System Parameter Upper-Case Prefix

Description	Converts any filenames in an OPEN statement to upper case when scanning a prefix file for a key matching the filename. Use this to simplify SQL migrations where all filenames must be in upper case.
Default	<i>Off.</i> ProvideX does not convert case when scanning a prefix file for a match. Normally, the keys in a prefix file are case-sensitive.

'PW'= System Parameter Password Character for Multi-Line

Description	Assigns the password character to be displayed in multi-lines to mask input denoted as a password. Use the ASCII value of your character; e.g., SET_PARAM 'PW'=42 will set the password character to "*".
Default	'PW'=36 or 'PW'=ASC("\$").

'PZ' System Parameter Suppress Program Size Warning

Description	Stops warnings from ProvideX regarding programs larger than 64K.
Default	<i>Off.</i> Warning will be sent when attempting to SAVE a program larger than 64K.

'Q_ '= System Parameter

Lowest Task Priority

Description *PVX Windows/WindX Only.* Assigns the lowest task priority level. When an application exceeds the value of the parameter '**WI** Windows Instruction Count, p.694, its priority level is decremented until it reaches the value of the '**Q_**' parameter.

Default 'Q_ '=2.



Note: Three parameters ('**Q_**', '**Q ^**', and '**QF**') control the priority of a task, primarily to balance the load in a client-server environment. PVXWIN32 . EXE supports five levels (range: lowest 0 to highest 4). The current priority level is stored in **TCB(91)**. Refer to the **TCB() Function, p.534**.

'Q ^ '= System Parameter

Highest Task Priority

Description *PVX Windows/WindX Only.* Assigns the highest task/thread priority level. The priority level is always reset to the '**Q ^**' value anytime a **WAIT** is executed or when terminal input is requested.

Default 'Q ^ '=2

'QD '= System Parameter

Windows Queue Display

Description Controls amount of time (in seconds) a session waits before forcing a check of the Windows Message Queue for keyboard/mouse activity. This parameter is only in affect when '**!W**' (WindX keyboard synchronization) is enabled and only while a task is executing code that does not issue an **INPUT** or **OBTAIN** within the delay period.

The '**QD**' parameter is a client-side setting that must be set after a WindX connection has been established via: **EXECUTE "[wdx]SET_PARAM 'QD'=nnn"**.



Note: During the delay period, ProvideX will not respond to **Ctrl - Break** or process mouse events, which could be perceived as the application being non-responsive.

Default 'QD '=4

'QF' System Parameter*Task Priority Factor*

Description *PVX Windows/WindX Only.* Controls how frequently the priority level is decremented. Setting 'QF' to a higher value will force the application to run for a longer duration at the given priority level before switching to the next lower priority level or until the 'Q_' value is reached.

Default 'QF'=1

'QK' System Parameter*Quick Key Lookup*

Description Activates an improved algorithm for scanning the key tree in Keyed and EFF files.

Default *On.*

'QS' System Parameter*START, Not Initialized*

Description Assists in conversions. A **START** command which includes the name of a program to run at startup will only clear local variables (same as a **BEGIN**) and start the specified program. ProvideX will only re-initialize on a simple **START** issued from the console.

Default *Off.* A **START** command to re-initialize ProvideX will close all files and clear all variables.

See Also [START Directive, p.328.](#)

'QT' System Parameter*No Prompt in Command Mode*

Description Suppresses the prompt in Command mode.

Default *Off.* Generates a prompt in Command mode.

'RI' System Parameter*Round Multi-Line Inputs*

Description *PVX Windows/WindX Only.* Rounds data in formatted input fields (multi-lines). When this parameter is set, data that has more than the allowed number of decimals will be rounded when placed into a field.

Default *Off.* Data is truncated.

'RN'= System Parameter

Rounding Control

Description Controls when and how rounding will occur. Valid settings for 'RN' are as follows:

- 1 - On any assignment
- 2 - On any Add, Subtract
- 4 - On any Multiply, Divide, Power, or Modulus
- 8 - On any math function (**SIN()**, **COS()**, **LOG()**, etc.)
- 64 - On all intermediate values in an expression
- 128 - Before any numeric data is written to file
- 256 - At the end of any expression
- 512 - Before all numeric compares

These values are *additive*; e.g., to set 4 + 128 + 512, use SET_PARAM 'RN' =644.

Default 'RN'=1 (on any assignment).

See Also 'RS' Round STR(), p.684.

'RP' System Parameter

Raw Print for *WINDEV*

Description *PVX Windows/WindX Only.*

On. To use the direct-to-spooler interface for *WINDEV* **Raw Print Mode**, p.756.

Off. To use the old *pass through* method of printing with *WINDEV*.



Warning: We recommend that you return to old *WINDEV* mode only if your client has problems printing to *WINDEV*.

Default *On.*

'RR' System Parameter

Reset on RUN

Description Issues a **RESET** command whenever a **RUN** statement is executed.

Default *Off.* See normal behaviour for the **RESET Directive**, p.288, and the **RUN Directive**, p.294.

'RS' System Parameter

Round STR()

Description Sets rounding for the STR() function.

Default *Off.* See normal behaviour for the **STR() Function**, p.525.

'SB' System Parameter*Self-Block Extracts*

Description *PVX UNIX/Linux Only.* Prevents a ProvideX process from extracting a record from a file if it already has that record extracted on another channel.



Note: Under Windows, this behaviour is the standard and is unchangeable. Extracts always block within the same process.

Default *Off.* Extracts are not blocked by other extracts within the same process.

'SC' System Parameter*Show Cursor*

Description *On.* To keep the cursor visible during processing.
Off. The cursor is hidden during processing. The cursor is restored for any user input.

Default *On.* For legacy DOS systems, the default is *Off.*)

'SD' System Parameter*Subdirectory Slash*

Description Appends a trailing slash to the name of a subdirectory entry returned when reading a directory.

Default *Off.* No trailing slash is returned with the subdirectory entry.

'SF' System Parameter*Short Form Variables*

Description Sets the compiler to restrict variable names to a single letter or a letter followed by a number. The **SHORT_FORM Directive**, *p.325*, can also be used to set **'SF'**. Use either **SET_PARAM -'SF'** or the **LONG_FORM Directive**, *p.201*, to cancel **'SF'**.



Note: System parameters **'LF'** and **'SF'** have reciprocal values and are directly related. Any change to **'SF'** results in an opposite change to **'LF'** and vice-versa.

Default *Off.* The compiler allows long variable names. Spaces are required between directives and variable names.

'SK' System Parameter

Shrink Keyed Files

- Description** Shrinks Keyed files when executing the directives **PURGE** or **REFILE**. ProvideX returns the freed space to the OS.
- Default** *Off.* By default, ProvideX simply re-initializes the file header information when it encounters a **PURGE/REFILE** directive for a data file (Keyed or Direct). This means that if you accidentally **PURGE/REFILE** such a data file, the ProvideX Keyed/Direct file key reconstruction utility (*UFAR) can salvage most, if not all, of the information originally stored in the file.
- See Also** [PURGE Directive, p.263](#),
[REFILE Directive, p.278](#).

'SL'= System Parameter

Save Command Lines

- Description** Assigns the number of command input lines to preserve internally. Range: 4 to 100.
- Default** 'SL'=32


'SP' System Parameter

Set Printer Default

- Description** Sets the ProvideX default printer to *serial printer* mode. For overstrike data, overlay characters will print on top of existing characters at their positions on the print line.
- Default** *Off.* Sets the printer to *line printer* mode. For overstrike data, overlay characters *replace* existing characters on the print line.

'SR' System Parameter

Small Reads

- Description** Attempts to optimize performance when reading fixed length record files over slower networks by reading smaller portions of the data record rather than the defined record size. If the first "short read" is unable to read the entire record then a second read is performed to retrieve the remaining portion of the record.
-  **Note:** This is used to improve performance in peer-to-peer networks where records are not filled to their maximum size.
- Default** *Off.* Reads the entire record in one read operation.

'SS' System Parameter

Check Structure on Save

Description	Checks <i>modified</i> programs for the logical integrity of some decision/loop structures and warns about any errors before saving (SAVE Directive, p.295).
Default	<i>Off</i> .

'SV'= System Parameter

Generate for Older Version

Description	Generates object code for compatibility with SBB.
Default	'SV'=1, the current program's object-version level.

'SW'= System Parameter

Scroll Wheel

Description	Sets default scroll wheel behaviour. <ul style="list-style-type: none"> 0 - Scroll wheel support (all events go to parent window) 1 - Scroll only if the control has focus (mouse can hover on or off control) 2 - Scroll only if the control has focus (mouse <i>must</i> be hovered over control) 3 - If control does not have focus, then scroll when mouse hovers over this control (otherwise, follow #1) 4 - If control does not have focus, then scroll when mouse hovers over this control (otherwise, follow #2)
Default	'SW'=1

'SZ'= System Parameter

Maximum Memory Size for Session

Description	Defines the maximum memory (in kilobytes) that the session is allowed to use. This value can also be changed via the START Directive, p.328 .
Default	'SZ'=32000. For older Windows versions of ProvideX, this value is 1024 or the maximum memory available, whichever is less. For older UNIX/Linux versions, this value is 512.



Note: Under Windows, the maximum setting is 32000. Under UNIX / Linux the limit has been increased from 512 to 32000. If the '**IZ**' parameter is set, the '**SZ**' parameter is ignored; see '[IZ](#)'=, [p.670](#).

'TA' = System Parameter Turbo Mode Acknowledgement

Description Defines the amount of data (in Kilobytes) that will be sent to the thin-client before forcing turbo mode acknowledgement. This has been shown to help slower speed connections; e.g., when printing across WANs.

Default 'TA'=0. No forced acknowledgement.

See Also 'TU' System Parameter, p.691.

'TB' System Parameter Toolbar Size

Description Calculates toolbar width/coordinates based on the number of columns in the window. The toolbar size will be the actual number of columns found in the window, which is equal to the value returned by MXC (0).

Default *Off*. Toolbar width and coordinates are calculated based on 80 columns, regardless of the number of columns in the window.

'TC' = System Parameter Tip Colour

Description Assigns the fill colour for the "Tip" window. If the colour code is in the range from 0 to 7 then the respective colour is used as fill. If the colour code is in the range from 8 to 15, then 50% of the colour is used. Supported options include:

0 - Current	8 - Dark Gray
1 - Light Red	9 - Dark Red
2 - Light Green	10 - Dark green
3 - Light Yellow	11 - Dark Yellow
4 - Light Blue	12 - Dark Blue
5 - Light Magenta	13 - Dark Magenta
6 - Light Cyan	14 - Dark Cyan
7 - White	15 - Gray

Default 'TC'=0. Current Windows default tip colour, specified in the user's desktop. This will match other Windows applications on the user's system.



Note: In earlier versions, 0 was the colour code for Black and the default setting was 11 for Yellow.

'TH' System Parameter

Thousands Separator

Description Assigns the thousands separator/character for use in formatted numeric data IO. Use the ASCII value of the character. For WindX environments, this parameter must be set on both the client and the server. In order for this format to be applied in numeric masking, the mask must be identified as numeric using the # character (see [Numeric Format Masks, p.814](#)).

Default 'TH'=44 or 'TH'=ASC(",") for the comma.



Note: 'TH' is used if I/O is formatted in **INPUT**, **OBTAIN**, **PRINT** and the **STR()** function (and ignored for unformatted I/O). It is always ignored by the **NUM()** function and when using **WRITE**, **READ**, **FIND** or **EXTRACT** directives in converting numeric data.

'TL' System Parameter

LIKE Emulates Thoroughbred

Description Sets the **LIKE Operator** to use a simplified pattern match which emulates the behaviour of **LIKE** in Thoroughbred for conversions from Thoroughbred. This pattern matching is:

- mask character: the complete string on the left must match
- character '*' in the mask stands for any number of characters
- character '?' in the mask stands for any single character

Default *Off.* **LIKE** uses the same pattern matching as **MSK()**, [p.486](#).

Example "ABCDEF" matches "*DEF", "ABC???", "*CD*" but not "ABC" or "?.".

'TN' System Parameter

Strip Trailing Nulls

Description Strips trailing nulls from the end of **READ RECORDS**. This parameter only applies to FLR-formatted (fixed-length record) **KEYED** files.

Default *Off.* Trailing nulls are not stripped.

'TT' = System Parameter

Timed Trace

Description Sets trace output prefixed by the current time (in seconds). Parameter codes allow control over where a line wraps based on the record size of the output file, output of the full program path name, and output of the line number only (no code). Valid settings for 'TT' are as follows:

- 0 - No output
- 1 - Trace line prefixed with time value followed by a space.
- 2 - Trace line prefixed with full program path. Delimiter becomes pipe, “|”
- 4 - Only the statement number is output (Program line is suppressed).
- 8 - Trace line prefixed with stack depth (Parameter 'BO' adjusted)

These values are *additive*; e.g., to set **2** and **4**, use SET_PARAM 'TT'=2+4.

Default 'TT'=0. No output.

'TU' System Parameter

Thin-Client Turbo Mode

Description Gains efficiency of throughput by not requiring the thin-client to acknowledge messages for directives/functions that have no return value (e.g., a **WRITE** for a graphical control). This improves speed in **NOMADS**.



Note: This parameter is set on the host/server. Also, when 'TU' is *on*, you lose error detection capability at the host end — errors are reported locally (e.g., on the WinDX client). Trouble-shoot by changing the application logic or by turning 'TU' *off*.

Default *Off*. Each tokenized message sent by the server to the client is acknowledged and errors are reported to your server, guaranteeing that your application and thin-client are fully synchronized (but at some cost to overall transmission speed).

See Also 'TA' = System Parameter, p.689.

'TX' System Parameter Default String-Template Field Separator

Description Sets BBx-style string templates that use the (*) format to default to the line feed character (\$0A\$) as a separator rather than \$8A\$.

Default *Off*. BBx-style string templates default to \$8A\$ as the separator.

'UL' System Parameter *Un-Numbered Line Assignment*

Description Automatically assigns line numbers to statements when loading external ASCII-based program files into ProvideX. This parameter simplifies the use of external editors for writing programs. When 'UL' is turned *off*, the line numbers assigned will be based on the 'AI' parameter setting.

Default *On*. Assigns numbers based on the line sequence of the source file (starting at 0001).

'UM' System Parameter *Upper Memory Blocks*

Description *Included for completeness only*. Sets ProvideX to use Upper Memory Blocks to increase memory space (legacy DOS systems).

Default *On*, if UMBs exist on the machine.

'VC' System Parameter *VT100 Cursor Mode Line Wrap*

Description Holds the cursor at column 80 until subsequent print data follows. This resolves an issue with wrapping at column 80 on VT100-type terminals such as a Linux console.

Default *Off*. Assumes terminal wraps when printing at column 80.

'VM' System Parameter *Direct Memory Addressing*

Description *Included for completeness only*. Sets all screen I/O to use direct memory addressing (legacy DOS systems).

Default *Off*. Screen I/O uses BIOS calls

'VP'= System Parameter *Variable Pitch*

Description Assigns the average character width for **PRINT @(x,y)** positioning with proportional fonts. Use the ASCII value of your character.

Default 'VP'=48 or 'VP'=ASC("0") for zero.

'VR' = System Parameter

Verify Read

Description Verifies all data file reads by re-reading and comparing. The values set in 'VR' define the number of retries before a data read error is generated.



Note: 'VR' can be helpful in tracking down hardware-related data corruption problems. (There is some impact on system performance.) **TCB()** returns the following values for this parameter:

TCB(63)- Number of reads verified

TCB(65)- Number of reads mis-compares.

Default 'VR'=0

See Also [TCB\(\) Return Task Information, p.534.](#)

'VW' = System Parameter

Verify Write

Description Verifies all data file writes by re-reading after a write to check data integrity. The value set in 'VW' defines the number of retries before a data write error is generated.



Note: 'VW' can be helpful in tracking down hardware-related data corruption problems. (There is some impact on system performance.) **TCB()** returns the following values for this parameter:

TCB(63)- Number of writes verified

TCB(65)- Number of writes mis-compares.

Default 'VW'=0

See Also [TCB\(\) Return Task Information, p.534.](#)

'WB' System Parameter

WindX BREAK Recognition

Description *On.* Enables WindX **BREAK** recognition.
Off. Disables WindX **BREAK** recognition.

Default *On.*

'WD' = System Parameter*Defer File Writes*

Description Defers all file writes until *nnn* updates have been done, the file is closed, or an **INPUT** is requested from channel #0 (terminal). The **'WD'** parameter only affects Keyed file I/O. Performance can be improved in single user environments if the **'WD'** parameter is set high.

Default **'WD'=10000** in a single user environment under Windows; otherwise, **'WD'=100**.

Example `SET_PARAM 'WD'=10000`

'WF' System Parameter*Force Windows Screen Update*

Description *PVX Windows/WindX Only.* Forces an update of the Windows screen whenever a change is made by the program.

Default *Off.* The screen is updated on **INPUT**, **WAIT**, or after processing the number of instructions set by the **'WI'** parameter.

'WH' = System Parameter*Delay Retry: Locking File Headers*

Description *All Platforms Except UNIX.* Defines the number of tenths of a second to delay between retries in locking file headers.

Default **'WH'=0** (zero tenths of second).

'WI' = System Parameter*Windows Instruction Count*

Description *PVX Windows/WindX Only.* Sets the number of instructions to be executed before passing control to the operating system.



Note: The overall number of instructions between priority level switching is based on an exponential formula using the values of the **'WI'** and **'QF'** parameters. See system parameters **'Q_'**, **'Q ^'** and **'QF'** [Task Priorities, p.682](#), which control task priority.

Default **'WI'=1000**

See Also [TCB\(\) Return Task Information, p.534](#); i.e., TCB(91).

'WK' System Parameter*Keep Window*

Description Prevents **'WINDOW'** and **'DIALOGUE'** boxes from being automatically dropped when you use a **BEGIN** (or **END** in Command mode) or a **CLOSE**.

Default *Off*.

'WL' System Parameter*Use Write Locks*

Description *PVX UNIX/Linux Only.* Temporarily blocks all other access for both **READ** and **WRITE** directives to a Keyed data file. (For that split second, only one access will be allowed.).

Default *Off.* ProvideX allows multiple simultaneous **READ** accesses to a Keyed file but only one **WRITE** access.

See Also [WRITE Add/Update Data in File, p.383](#) and [READ Read Data from File, p.271](#).

'WP' System Parameter*Wait for Pipe on Close*

Description Determines whether or not to wait until a child process completes. When **'WP'** is on, and a pipe is closed, ProvideX will wait until the child process has terminated.



Note: If **'WP'** is on, and the operation that the child was started for does not complete, then ProvideX will not return. A **Ctrl - Break** may terminate the child, depending on the child's processing of a **BREAK**.

When **'WP'** is off, and a pipe is closed, ProvideX will not wait for the child to complete. This can result in a child process that has not had its exit status checked (shown as "defunct"). ProvideX will automatically check and clear out these defunct processes whenever **A**) it opens another pipe, **B**) a **WAIT** directive is encountered, or **C**) the current ProvideX session is terminated.

Default *Off*.

'WT'= System Parameter*Number of Retries*

Description Assigns the number of retries ProvideX makes internally for busy records/ files. The retries are performed at one second intervals.

Default **'WT'=2**

'WZ' System Parameter*WindX ZLib Compression*

Description	Sets the minimum packet size (in bytes) for compression during WindX transmission. Values can range from 0 to 32767. Setting the value to 0 disables ZLib compression completely. <i>This parameter must be set on the host/server.</i>
Default	'WZ'=512

'XC' System Parameter*WindX Continues After TCP Error*

Description	Sets ProvideX to ignore TCP write errors to WindX. This allows a process to continue running and outputting to WindX, even while the WindX client is no longer connected. The process should continue until a READ/INPUT request; then, the error is reported and ProvideX terminates
Default	<i>Off.</i> WindX generates the error and cancels the process.

'XF' System Parameter*Extended File Channels*

Description	Sets extended file mode where the channels range from 1-32767 for local files and 32768-65000 for global files. Channel 0 <i>zero</i> is the console or terminal.
Default	<i>Off.</i> Channels 1 to 63 are used for local files and 64 to 127 are used for global files.



Note: Before you turn the 'XF' parameter *off*, make sure that none of your currently open files have extended file numbers (which are inaccessible when you SET_PARAM - 'XF').

'XI' System Parameter*Extract Ignore*

Description	Allows extracted records to be read by other ProvideX processes. The blocking of the extracted record is limited to EXTRACT , WRITE and REMOVE .
Default	<i>Off.</i> The EXTRACT of a record blocks all other file I/O on that record, <i>including READ</i> .
See Also	EXTRACT Read and Lock Data, p.126 and WRITE Add/Update Data in File, p.383 .

'XL' System Parameter*Obsolete*

Description	<i>Deprecated.</i> Use the ' XI ' System Parameter, p.696 .
-------------	--

'XS' = System Parameter**Extended Memory (KB)**

Description	<i>Included for completeness only.</i> Assigns the amount of extended memory used as program swap space (legacy DOS systems). If the value exceeds the available extended memory, it will be adjusted to the maximum allowed.
Default	'XS'=0

'XT' System Parameter**ProvideX Exits to OS**

Description	<i>On.</i> To terminate ProvideX and return to the OS when your application returns to Command mode (on end of program, a detected error, or an Escape). <i>Off.</i> To return and stay in Command mode on end of program, error, or Escape.
Default	<i>On</i> , if there is a lead program on the startup ProvideX command. Otherwise, it is <i>off</i> .



Note: If the **ESCAPE** directive is used in a program, this system parameter will be automatically reset to prevent session termination.

'ZP' System Parameter**Accept Zero-Length Programs**

Description	Ignores Error #18: Program not loaded/Invalid program format. As of Version. 4.20, empty program files are considered to be valid.
Default	<i>Off.</i> RUN, CALL, LOAD or PERFORM commands with an empty program file generate an Error #18.

'!9' System Parameter**Sage MAS 90 Date Format**

Description	<i>Sage MAS 90 Only.</i> Converts incoming and outgoing dates for an ODBC connection to Sage MAS 90 format automatically.
Default	<i>Off.</i> No automatic conversion is performed.

'!B' = System Parameter**Set Break Character**

Description	<i>PVX UNIX/Linux Only.</i> Assigns the break character.
Default	'!B'=3

'!D' System Parameter *Numeric Separators: Legacy Mode*

Description Resets to the original method for handling the decimal point and the thousands separator prior to ProvideX 4.12.

Default *Off.* See the **'DP'= System Parameter, p.662**, and the **'TH'= System Parameter, p.690**, for the current method of handling these values.

'!F' System Parameter *Obsolete*

Description *Included here for completeness only.*

'!I' System Parameter *NOMADS Input Queue*

Description Activates the NOMADS input queue for macro playback.

Default *Off.* To de-activate the queue.

'!K' System Parameter *Descending Key Logic (Legacy)*

Description To use legacy descending key logic. This is included in ProvideX only to help programmers who relied on the legacy logic's incorrect behaviour. Issue on **READ(c,KEY=)** for primary key with descending segments (resolved Version 4.11).

Default *Off.* ProvideX standard logic is used.

'!Q'= System Parameter *ODBC SQL Display*

Description Displays SQL statements to assist in the troubleshooting of external database related problems. If using the WindX client, then the SQL statement will appear on the client. Valid values are as follows:

- 0 - *Off.* Hidden. *Default.*
- 1 - Message box appears with generated SQL statement.
- 2 - SQL statement is sent to program trace window (if open).

'!R'= System Parameter *For Internal Use Only*

Reserved *Included here for completeness only.*

'!S' System Parameter *Suppress Error Flags on Serial Save*

Description Will not flag lines with errors when saving to serial files. Enabling '!S' removes the question mark immediately following the line number on statements with errors.

Default *Off.* Errors are flagged.

'!T' System Parameter *'DP' or Decimal for Numerics*

Description Accepts either 'DP' or the true decimal point character (".") as a decimal point for numeric data input.

Default *Off.* Hidden.

'!U'= System Parameter *For Internal Use Only*

Reserved *Included here for completeness only.*

'!V' System Parameter *I'm a Service*

Description *PVX Windows/WindX Only.* When enabled, ProvideX will quietly process any Windows termination signals received from the operating system as if it were running as a background service.

Default *Off.*

'!W' System Parameter *WindX Keyboard Synchronization*

Description Forces ProvideX and WindX to maintain keyboard synchronization in order to prevent type-ahead buffer loss.

Default *Off.*



Note: Users of *NTHost/*NTSlave will need to upgrade both the server side and WindX clients to properly utilize the WindX type ahead improvements controlled with '!W'.
Also: Only set this parameter on the server side of a client-server connection.

'!X' System Parameter*I/O Crossover*

Description Map **READ** and **WRITE** requests to channel 0 internally to **INPUT** and **PRINT** respectively (to assist with BBx conversion).

Default *Off.*

'*K' System Parameter*Obsolete*

Description *Included here for completeness only.*

'*L' System Parameter*Obsolete*

Description *Included here for completeness only.*



7

Control Object Properties

Align	ColumnPixels	ItemCount	OnFocusCols	Sort
Align\$	ColumnSizeLock	ItemNeededCtl	OnFocusLines	Sort\$
Auto	ColumnsWide	ItemNeededFrom	OnFocusCtl	SortCaseSensitive
AutoComplete\$	ColumnWidth	ItemNeededTo	OverlapEnabled	SortColFmt\$
AutoCTL	CurrentCellColour\$	ItemState	Parent	SortOnHdrClick
AutoScale	CurrentColumn	ItemTag\$	PointText\$	SortStyle
AutoSequence	CurrentItem	ItemText\$	PrefixData	StateBitmaps\$
AutoState	CurrentPoint	JoinColumns	Proportions2M	SwapEnabled
AutoTrack	CurrentRow	JoinRows	ProportionDW	TabMode
AutoValue\$	CurrentSet	Key\$	ProportionHW	Tbl\$
BackColour\$	CtlName\$	LabelLocation\$	RangeMax	TblWidth
BackHighlight1\$	DraggedColumn	Left	RangeMin	Text\$
BackHighlight2\$	DraggedRow	LeftBorder	RangeText\$	TextColour\$
BigJump	DroppedOn	LegendLocation\$	Resizable	TickPerUnit
Bitmap\$	DroppedOnColumn	LegendText\$	RightBorder	TickPixels
BitmapPosition	DroppedOnRow	Len	Row	Tip\$
BitmapPosition\$	Edit	Line	RowData\$	Title1\$
BottomBorder	Enabled	LineColour\$	RowHeight	Title2\$
BottomLeftTick\$	EnterMode	Lines	RowHiLight	Top
Calendar\$	Eom\$	LoadIOLIST\$	RowPixels	TopBorder
CascadeState	ExcelStyle	LoadList\$	RowsHigh	TopLeftTick\$
CellFormat\$	Expanded	Lock	Scroll	TrackColour\$
CellHiLight	FaceColourBack\$	LockColumns	ScrollWheel	Uppercase
CellImpliedDecimal	FaceColourBottom\$	LockRows	SelectColumn	Value
CellLeft	FaceColourLeft\$	MarginBottom	SelectCount	Value\$
CellTag\$	FillColour\$	MarginLeft	SelectedChildren	Visible
CellTbl\$	Fmt\$	MarginRight	SelectIndex	Width
CellTblWidth	Focus	Margins\$	SelectItem	XAxisLocation\$
CellTip\$	Font\$	MarginTop	SelectLength	XAxisTitle\$
CellTop	Footer\$	MaxValue	SelectOffset	YAxisLocation\$
CellType\$	GenerateScrollEom	MenuColumn	SelectRow	YAxisTitle\$
CellTypeList\$	Height	MenuCtl	SelectStateMask	ZAxisLocation\$
Children	HoverColour\$	MenuRow	SelectText\$	ZAxisTitle\$
Col	hWnd	MouseOver	SelectValue\$	_PropList\$
Cols	Id	Msg\$	Sep\$	_PropSep\$
Column	ImageCount	MultiSelect	SepLoad\$	_PropValues\$
Column\$	ImpliedDecimal	NotifyExpand	SignalOnExit	
ColumnClicked	IndexMode	Nul\$	SignalOnly	
ColumnNames\$	InsDelEnabled	NumPoints	SkipLockedCells	
	Item	NumSets	SmallJump	

Overview

Various properties of **Graphical Control Objects** in ProvideX (button, drop box, scrollbar ...) can be referenced and modified *dynamically* using a control's assigned CTL value (*ctl_id*) followed by the apostrophe operator and one of the associated *property names* (listed above). This chapter describes the various properties and discusses how they are used to define controls in ProvideX.



Note: In this reference, some properties are denoted with **\$ dollar signs** to indicate that they represent string values; e.g., 'Msg\$ or 'Tip\$.

For a complete list of graphical control objects and their properties, see [Graphical Control Objects, p.703](#). The [Properties List, p.709](#), provides a complete alphabetically- arranged list of valid property names and their definitions. Special property groupings (for extended GUI functionality) are described in the section [Compound Properties, p.728](#).

Using Property Names

As mentioned earlier, access to control object properties is provided via the apostrophe operator. (For syntax, refer to the [Apostrophe Operator, p.823](#)).

For example, a button's location, size, text, and colour would be represented by the properties **Height**, **Font\$**, **Text\$**, **TextColour\$**, etc. If the variable `MyButton` contained the CTL value associated with a button, you could change its text as follows:

```
MyButton'Text$ = "Hit me now"
```

Other common properties include:

'Col	Column
'Line	Line
'Cols	Width of the control
'Lines	Height of the control
'Tip\$	Tip for the control
'Msg\$	Message line for the control
'Fmt\$	Format mask for control
'TextColour\$	Text Colour
'Value\$	Current value/state of control.



Note: While programs can access or update property values, properties cannot be specified as the target for any file I/O or **CALL** parameter lists.

Generally, numeric properties are type insensitive; i.e., a property such as 'Line returns (or receives) a number. If desired, you can access the same value using the property 'Line\$. This is also true for string properties, assuming that they only return numeric values.

Some properties return different values based on the type of reference you make. For example, most colour properties return a text description of the RGB colour when accessed as a string, or a 24-bit colour number when accessed as a numeric.

ProvideX also supports objects that are external to ProvideX this chapter does not deal with the properties (and methods) that apply to them. COM and OOP objects/controls are described under the [Apostrophe Operator, p.823](#).

Graphical Control Objects

Graphical control objects are used in ProvideX applications to display information, input data, and handle event processing. These controls can be created using specific directives or designed/produced in NOMADS, the ProvideX GUI-based application development system. Refer to the *ProvideX NOMADS Reference* for further information.

The following *control object types* are supported in ProvideX:

BUTTON	GRID	RADIO_BUTTON	VARLIST_BOX
CHART	LIST_BOX	TREE_VIEW	V_SCROLLBAR
CHECK_BOX	LIST_VIEW	TRISTATE_BOX	H_SCROLLBAR
DROP_BOX	MULTI_LINE	VARDROP_BOX	

This section provides cross-references to corresponding directives and lists all of the properties used to define and manipulate each of the specific object types. The list supplied immediately below each object heading is linked to descriptions under the full [Properties List, p.709](#).

Properties that define *extended attributes* are shown in *italics*. They represent a category of attributes that cannot be accessed within a directive (via **FMT=** or **OPT=**); for example, the majority of *cell attributes* for *grids* are defined/set using properties. See also [Compound Properties, p.728](#).

BUTTON

BackColour\$	Focus	<i>Left</i>	SignalOnly	<i>_PropList\$</i>
BitmapPosition	Font\$	Line	Text\$	<i>_PropSep\$</i>
Col	<i>Height</i>	Lines	TextColour\$	<i>_PropValues\$</i>
Cols	<i>HoverColour\$</i>	MenuCtl	Tip\$	
CtlName\$	<i>ImageCount</i>	Msg\$	<i>Top</i>	
Enabled	<i>hWnd</i>	OnFocusCtl	Visible	
Eom\$	Key\$	<i>Parent</i>	Width	

A button object is usually designed to send a signal to the application when selected by a mouse click. The signal typically indicates that the user wants to end a function or initiate a new function. For more information on **BUTTON** controls, refer to the [BUTTON Directive, p.34](#).

CHART

<i>AutoScale</i>	<i>FaceColourLeft\$</i>	<i>MarginBottom</i>	<i>RangeMax</i>	<i>Visible</i>
<i>BackColour\$</i>	<i>Fmt\$</i>	<i>MarginLeft</i>	<i>RangeMin</i>	<i>Width</i>
<i>Bitmap\$</i>	<i>Font\$</i>	<i>MarginRight</i>	<i>RangeText\$</i>	<i>XAxisLocation\$</i>
<i>BitmapPosition\$</i>	<i>Footer\$</i>	<i>Margins\$</i>	<i>SelectIndex</i>	<i>XAxisTitle\$</i>
<i>Col</i>	<i>Height</i>	<i>MarginTop</i>	<i>Sep\$</i>	<i>YAxisLocation\$</i>
<i>Cols</i>	<i>hWnd</i>	<i>MenuCtl</i>	<i>SepLoad\$</i>	<i>YAxisTitle\$</i>
<i>CurrentPoint</i>	<i>IndexMode</i>	<i>NumPoints</i>	<i>TextColour\$</i>	<i>YAxisLocation\$</i>
<i>CurrentSet</i>	<i>LabelLocation\$</i>	<i>NumSets</i>	<i>Tip\$</i>	<i>ZAxisTitle\$</i>
<i>CtlName\$</i>	<i>Left</i>	<i>Parent</i>	<i>Title1\$</i>	<i>_PropList\$</i>
<i>Enabled</i>	<i>LegendLocation\$</i>	<i>PointText\$</i>	<i>Title2\$</i>	<i>_PropSep\$</i>
<i>Eom\$</i>	<i>LegendText\$</i>	<i>Proportions2M</i>	<i>Top</i>	<i>_PropValues\$</i>
<i>FaceColourBack\$</i>	<i>Line</i>	<i>ProportionDW</i>	<i>Value</i>	
<i>FaceColourBottom\$</i>	<i>Lines</i>	<i>ProportionHW</i>	<i>Value\$</i>	

The chart control is used to create illustrations for an application. A chart is usually designed to be a *display only* object that requires no user interaction. For more information on this control, refer to the [CHART Directive, p.43](#). For information on applying properties to individual chart elements (labels, legends), see [Chart Label Reference, p.734](#).

CHECK_BOX

<i>BackColour\$</i>	<i>Focus</i>	<i>Left</i>	<i>SignalOnly</i>	<i>Value\$</i>
<i>BitmapPosition</i>	<i>Font\$</i>	<i>Line</i>	<i>Tbl\$</i>	<i>Visible</i>
<i>Col</i>	<i>Height</i>	<i>Lines</i>	<i>Text\$</i>	<i>Width</i>
<i>Cols</i>	<i>HoverColour\$</i>	<i>MenuCtl</i>	<i>TextColour\$</i>	<i>_PropList\$</i>
<i>CtlName\$</i>	<i>ImageCount</i>	<i>Msg\$</i>	<i>Tip\$</i>	<i>_PropSep\$</i>
<i>Enabled</i>	<i>hWnd</i>	<i>OnFocusCtl</i>	<i>Top</i>	<i>_PropValues\$</i>
<i>Eom\$</i>	<i>Key\$</i>	<i>Parent</i>	<i>Value</i>	

A check box object is designed to be toggled between *two* states: **ON** to *check* the option or **OFF** to *uncheck* it. For more information on the **CHECK_BOX** control, refer to the [CHECK_BOX Directive, p.47](#).

DROP_BOX

<i>Auto</i>	<i>Eom\$</i>	<i>Key\$</i>	<i>ScrollWheel</i>	<i>Top</i>
<i>BackColour\$</i>	<i>Focus</i>	<i>Left</i>	<i>Sep\$</i>	<i>Value</i>
<i>Col</i>	<i>Font\$</i>	<i>Line</i>	<i>SepLoad\$</i>	<i>Value\$</i>
<i>Cols</i>	<i>Height</i>	<i>Lines</i>	<i>SignalOnExit</i>	<i>Visible</i>
<i>CurrentItem</i>	<i>hWnd</i>	<i>MenuCtl</i>	<i>Tbl\$</i>	<i>Width</i>
<i>CtlName\$</i>	<i>Item</i>	<i>Msg\$</i>	<i>TblWidth</i>	<i>_PropList\$</i>
<i>DroppedOn</i>	<i>ItemCount</i>	<i>OnFocusCtl</i>	<i>TextColour\$</i>	<i>_PropSep\$</i>
<i>Enabled</i>	<i>ItemText\$</i>	<i>Parent</i>	<i>Tip\$</i>	<i>_PropValues\$</i>

This control is used to provide a drop-down list of elements from which users can make a selection. A drop box takes a smaller amount of space on the screen than a comparable list box. For more information on this control, refer to the [DROP_BOX Directive, p.96](#).

GRID

Align	Column\$	<i>hWnd</i>	<i>OverlapEnabled</i>	<i>SortCaseSensitive</i>
Align\$	ColumnNames\$	<i>ImpliedDecimal</i>	<i>Parent</i>	<i>SortOnHdrClick</i>
Auto	ColumnPixels	<i>InsDelEnabled</i>	<i>Resizable</i>	<i>SortColFmt\$</i>
<i>AutoSequence</i>	ColumnSizeLock	<i>JoinColumns</i>	<i>RightBorder</i>	<i>SortStyle</i>
<i>AutoTrack</i>	ColumnsWide	<i>JoinRows</i>	<i>Row</i>	<i>SwapEnabled</i>
BackColour\$	ColumnWidth	Key\$	RowData\$	<i>TabMode</i>
<i>Bitmap\$</i>	CurrentColumn	<i>Left</i>	<i>RowHeight</i>	<i>Text\$</i>
BottomBorder	CurrentCellColour\$	<i>LeftBorder</i>	<i>RowHiLight</i>	<i>TextColour\$</i>
<i>BottomLeftTick\$</i>	CurrentRow	<i>Len</i>	RowPixels	<i>TickPerUnit</i>
CellFormat\$	CtlName\$	<i>Line</i>	RowsHigh	<i>TickPixels</i>
CellHiLight	<i>DraggedColumn</i>	<i>Lines</i>	<i>SelectColumn</i>	<i>Tip\$</i>
CellImpliedDecimal	<i>DraggedRow</i>	<i>LoadIOLIST\$</i>	<i>SelectCount</i>	<i>Top</i>
CellLeft	DroppedOnColumn	<i>LoadList\$</i>	<i>SelectIndex</i>	<i>TopBorder</i>
CellTag\$	DroppedOnRow	<i>Lock</i>	<i>SelectRow</i>	<i>TopLeftTick\$</i>
<i>CellTbl</i>	Enabled	<i>LockColumns</i>	<i>SelectText\$</i>	<i>TrackColour\$</i>
<i>CellTblWidth</i>	EnterMode	<i>LockRows</i>	<i>SelectValue\$</i>	<i>Uppercase</i>
CellTip\$	Eom\$	<i>MenuColumn</i>	<i>ScrollWheel</i>	<i>Value\$</i>
CellTop	ExcelStyle	<i>MenuCtl</i>	Sep\$	<i>Value</i>
CellType\$	FillColour\$	<i>MenuRow</i>	SepLoad\$	<i>Visible</i>
CellTypeList\$	Fmt\$	<i>Msg\$</i>	<i>SignalOnExit</i>	<i>Width</i>
Col	Focus	<i>MultiSelect</i>	<i>SkipLockedCells</i>	<i>_PropList\$</i>
Cols	Font\$	<i>Nul\$</i>	<i>Sort</i>	<i>_PropSep\$</i>
Column	<i>Height</i>	<i>OnFocusCtl</i>	<i>Sort\$</i>	<i>_PropValues\$</i>

The **GRID** control is used to create a table of cells in columns and rows; i.e., a spreadsheet input format. For more information, refer to the **GRID Directive**, p. 143. See also **Grid Property Access**, p. 734, **Multiple Selections**, p. 730, **Drag and Drop**, p. 733, and **Loading/Accessing by Row**, p. 733.

LIST_BOX

Auto	Fmt\$	<i>ItemText\$</i>	SelectCount	Value
BackColour\$	Focus	Key\$	SelectIndex	Value\$
<i>BackHighlight1\$</i>	Font\$	<i>Left</i>	<i>SelectItem</i>	<i>Visible</i>
<i>BackHighlight2\$</i>	<i>Height</i>	<i>Line</i>	Sep\$	<i>Width</i>
Col	<i>HoverColour\$</i>	<i>Lines</i>	SepLoad\$	<i>_PropList\$</i>
Cols	<i>hWnd</i>	<i>MenuCtl</i>	<i>SignalOnExit</i>	<i>_PropSep\$</i>
CurrentItem	<i>Item</i>	<i>MouseOver</i>	Tbl\$	<i>_PropValues\$</i>
CtlName\$	<i>ItemCount</i>	<i>Msg\$</i>	TblWidth	
DroppedOn	ItemNeededCtl	OnFocusCtl	TextColour\$	
Enabled	<i>ItemNeededFrom</i>	<i>Parent</i>	Tip\$	
Eom\$	<i>ItemNeededTo</i>	<i>ScrollWheel</i>	<i>Top</i>	

A list box displays a list of elements from which the users can make a selection. ProvideX supports different list box types: *Standard* and *Formatted* (see the **LIST_BOX Directive**, p. 178), **LIST_VIEW** (described below), and **TREE_VIEW** (described below). See also **Multiple Selections**, p. 730 and **Load on Demand**, p. 729.

LIST_VIEW

Auto	DroppedOn	<i>ItemNeededFrom</i>	ScrollWheel	Tip\$
BackColour\$	Enabled	<i>ItemNeededTo</i>	SelectCount	Top
<i>BackHighlight1\$</i>	Eom\$	<i>ItemText\$</i>	SelectIndex	Value
<i>BackHighlight2\$</i>	Fmt\$	Key\$	<i>SelectItem</i>	Value\$
Col	Focus	<i>Left</i>	Sep\$	Visible
Cols	Font\$	Line	SepLoad\$	Width
Column	<i>Height</i>	Lines	SignalOnExit	<i>_PropList\$</i>
ColumnClicked	<i>HoverColour\$</i>	MenuCtl	Sort	<i>_PropSep\$</i>
<i>ColumnsWide</i>	<i>hWnd</i>	<i>MouseOver</i>	<i>SortOnHdrClick</i>	<i>_PropValues\$</i>
<i>ColumnWidth</i>	<i>Item</i>	Msg\$	Tbl\$	
CurrentItem	<i>ItemCount</i>	OnFocusCtl	TblWidth	
CtlName\$	ItemNeededCtl	<i>Parent</i>	TextColour\$	

This control operates like a standard list box but provides for columnar lists with optional bitmaps. The format appears similar to the right-side pane of “classic” Windows Explorer. Details on the list view control are provided under the **LIST_BOX** Directive, *p.189*. See also **Load on Demand**, *p.729*, and **Multiple Selections**, *p.730*.

MULTI_LINE

Auto	Focus	Lines	ScrollWheel	Value
AutoComplete\$	Font\$	Lock	SelectLength	Value\$
BackColour\$	GenerateScrollEom	MenuCtl	<i>SelectOffset</i>	Visible
Calendar\$	<i>Height</i>	Msg\$	<i>SelectText\$</i>	Width
Col	<i>hWnd</i>	Nul\$	Sep\$	<i>_PropList\$</i>
Cols	<i>ImpliedDecimal</i>	OnFocusCtl	SignalOnExit	<i>_PropSep\$</i>
CtlName\$	Key\$	OnFocusCols	TextColour\$	<i>_PropValues\$</i>
Enabled	<i>Left</i>	OnFocusLines	Tip\$	
Eom\$	<i>Len</i>	<i>Parent</i>	<i>Top</i>	
Fmt\$	Line	<i>Scroll</i>	Uppercase	

Multi-lines provide a standard input field to display and enter one or more lines of text. For more information on the **MULTI_LINE** control, refer to the **MULTI_LINE** Directive, *p.215*.

RADIO_BUTTON

BackColour\$	Focus	Key\$	<i>Parent</i>	Value
BitmapPosition	Font\$	<i>Left</i>	SignalOnly	Value\$
Col	Height	Line	Tbl\$	Visible
Cols	<i>HoverColour\$</i>	Lines	Text\$	Width
CtlName\$	<i>hWnd</i>	MenuCtl	TextColour\$	<i>_PropList\$</i>
Enabled	<i>ImageCount</i>	Msg\$	Tip\$	<i>_PropSep\$</i>
Eom\$	Id	OnFocusCtl	<i>Top</i>	<i>_PropValues\$</i>

Radio buttons are used to control a variable between a series of preset states, offering one choice from a group of options. When one radio button is selected, it becomes activated (on) and all other related radio buttons are automatically reset (off). For more information on the **RADIO_BUTTON** control, refer to the **RADIO_BUTTON** Directive, *p.265*

TREE_VIEW

Auto	Eom\$	Key\$	ScrollWheel	Tip\$
AutoState	<i>Expanded</i>	<i>Left</i>	SelectCount	<i>Top</i>
BackColour\$	Fmt\$	Line	SelectedChildren	Value
CascadeState	Focus	LineColour\$	SelectIndex	Value\$
Children	Font\$	Lines	<i>SelectItem</i>	Visible
Col	<i>Height</i>	MenuCtl	<i>SelectStateMask</i>	Width
Cols	<i>hWnd</i>	<i>MouseOver</i>	Sep\$	<i>_PropList\$</i>
CurrentItem	<i>Item</i>	Msg\$	SepLoad\$	<i>_PropSep\$</i>
CtlName\$	<i>ItemCount</i>	<i>NotifyExpand</i>	SignalOnExit	<i>_PropValues\$</i>
DroppedOn	<i>ItemState</i>	OnFocusCtl	Sort	
Edit	ItemTag\$	<i>Parent</i>	<i>StateBitmaps\$</i>	
Enabled	<i>ItemText\$</i>	<i>PrefixData</i>	TextColour\$	

This control operates like a standard list box but appears as a tree-like structure with optional bitmaps. Each entry in a tree view consists of a series of strings or values separated by a delimiter like a directory structure. Details on the tree view list box control are provided under the [LIST_BOX Directive, p.192](#). See also [State Indicators](#) and [Multiple Selections, p.730](#).

TRISTATE_BOX

BackColour\$	Focus	<i>Left</i>	SignalOnly	Value\$
BitmapPosition	Font\$	Line	Tbl\$	Visible
Col	<i>Height</i>	Lines	Text\$	Width
Cols	<i>HoverColour\$</i>	MenuCtl	TextColour\$	<i>_PropList\$</i>
CtlName\$	<i>hWnd</i>	Msg\$	Tip\$	<i>_PropSep\$</i>
Enabled	<i>ImageCount</i>	OnFocusCtl	<i>Top</i>	<i>_PropValues\$</i>
Eom\$	Key\$	<i>Parent</i>	Value	

A tristate box is a check box in which the user can toggle between three states: On, Off, and a third choice. For more information, refer to the [TRISTATE_BOX Directive, p.344](#).

VARDROP_BOX

Auto	Font\$	Line	<i>SelectText\$</i>	Visible
BackColour\$	<i>Height</i>	Lines	Sep\$	Width
Col	<i>hWnd</i>	MenuCtl	SepLoad\$	<i>_PropList\$</i>
Cols	<i>Item</i>	Msg\$	SignalOnExit	<i>_PropSep\$</i>
CtlName\$	<i>ItemCount</i>	OnFocusCtl	TextColour\$	<i>_PropValues\$</i>
DroppedOn	<i>ItemText\$</i>	<i>Parent</i>	Tip\$	
Enabled	Key\$	ScrollWheel	<i>Top</i>	
Eom\$	<i>Left</i>	SelectLength	Value	
Focus	<i>Len</i>	<i>SelectOffset</i>	Value\$	

The [VARDROP_BOX](#) control operates like a standard drop box but will allow variable input. That is, the user can select any element from a list of items associated with the drop box but can also enter any other value. For more information on this control, refer to the [VARDROP_BOX Directive, p.354](#).

VARLIST_BOX

Auto	Font\$	Line	SelectText\$	Visible
BackColour\$	Height	Lines	Sep\$	Width
Col	hWnd	MenuCtl	SepLoad\$	_PropList\$
Cols	Item	Msg\$	SignalOnExit	_PropSep\$
CtlName\$	ItemCount	OnFocusCtl	TextColour\$	_PropValues\$
DroppedOn	ItemText\$	Parent	Tip\$	
Enabled	Key\$	ScrollWheel	Top	
Eom\$	Left	SelectLength	Value	
Focus	Len	SelectOffset	Value\$	

The **VARLIST_BOX** control operates like a standard list box but will allow variable input. That is, the user can select any element from a list of items associated with the list box but can also enter any other value. For more information on this control, refer to the [VARLIST_BOX Directive, p.360](#).

V_SCROLLBAR and H_SCROLLBAR

Auto	Enabled	Line	ScrollWheel	Width
BigJump	Height	Lines	Top	_PropList\$
Col	hWnd	MaxValue	Value	_PropSep\$
Cols	Key\$	Parent	Value\$	_PropValues\$
CtlName\$	Left	SmallJump	Visible	

The **V_SCROLLBAR** and **H_SCROLLBAR** controls are designed to create and manipulate vertical and horizontal scrollbars on the screen. For more information on this control, refer to the [V_SCROLLBAR Directive, p.365](#), and the [H_SCROLLBAR Directive, p.153](#).

Properties List

The various properties available for defining ProvideX common control objects are described below. Each property name cross-references to one or more control object(s), which are described in the section [Graphical Control Objects, p.703](#).

- Align** [GRID](#)
Text alignment. Valid numeric values include 1 (top-right), 2 (top-centre), 3 (top-left), 4 (right), 5 (centre), 6 (left), 7 (bottom-right), 8 (bottom-centre), 9 (bottom-left). *Default:* 4.
- Align\$** [GRID](#)
Text alignment. Valid text values include "TR" (top-right), "TC" (top-centre), "TL" (top-left), "R" (right), "C" (centre), "L" (left), BR (bottom-right), "BC" (bottom-centre), "BL" (bottom-left). *Default:* "TL".
- Auto** [DROP_BOX](#) [GRID](#) [H_SCROLLBAR](#) [LIST_BOX](#) [LIST_VIEW](#) [MULTI_LINE](#) [TREE_VIEW](#) [VARDROP_BOX](#) [VARLIST_BOX](#) [V_SCROLLBAR](#)
Signal on all changes: 1=True, 0=False. *Default:* 0.
- AutoComplete\$** [MULTI_LINE](#)
Automatic Completion of User Entries. For parameters, see [AutoComplete, p.218](#).
- AutoCTL** [MULTI_LINE](#)
Generates ctl_id to signal Auto-Complete. *ctl_id* to be generated by a [MULTI_LINE](#) to signal that the list of entries for the [AutoComplete](#) dropbox is to be loaded via the [AutoValue\\$](#) property. .
- AutoScale** [CHART](#)
Auto-scaling for text elements. 0=Off; 1=On. When set to On, chart text elements (labels, legends) are automatically resized to fit the given area. *Default:* 1. For more information on applying this property, see [Chart Label Reference, p.734](#).
- AutoSequence** [GRID](#)
Automatic row sequence. Assigns a column number that contains a row sequence number. ProvideX automatically fills in values of the cells with the appropriate row number. When rows are changed, the column containing the sequence number is automatically updated. (*Changing this property forces full redraw of the grid.*) *Default:* 0.
- AutoState** [TREE_VIEW](#)
Control automatic toggling of states. See [State Indicators, p.731](#).
- AutoTrack** [GRID](#)
Automatic Scrolling Control. Assigns *horizontal-vertical* scrolling modes *normal* (N), *scrolltracking* (S), and *joystick* (J) . Codes include: 0=N-N; 1=N-S; 2=S-N; 3=S-S; 4=N-J; 5=S-J . *Default:* 0. See [Scroll Modes](#) under the [GRID Directive, p.147](#).
- AutoValue\$** [MULTI_LINE](#)
List of entries to be loaded into auto-complete dropbox. See [AutoCTL](#).

- BackColor\$** [BUTTON](#) [CHART](#) [CHECK_BOX](#) [DROP_BOX](#) [GRID](#) [LIST_BOX](#) [LIST_VIEW](#) [MULTI_LINE](#) [RADIO_BUTTON](#) [TREE_VIEW](#) [TRISTATE_BOX](#) [VARDROP_BOX](#) [VARLIST_BOX](#)
Background colour. Valid colour names are listed under [Colour Properties, p.727](#).
Default: "DEFAULT".
- BackColor\$** *Background colour* – US spelling. See above.
- BackHilight1\$** [LIST_BOX](#) [LIST_VIEW](#)
Background colour, alternating lines. Valid colour names are listed under [Colour Properties, p.727](#). *Default:* "DEFAULT".
- BackHilight2\$** [LIST_BOX](#) [LIST_VIEW](#)
Background colour, alternating three lines. Valid colour names are listed under [Colour Properties, p.727](#). *Default:* "DEFAULT".
- BigJump** [H_SCROLLBAR](#) [V_SCROLLBAR](#)
Scrollbar big jump value. *Default:* 0.
- Bitmap\$** [CHART](#) [GRID](#)
Bitmap to be used as chart background, or in grid cell. Specify embedded (e.g., !Stop) or external (e.g., C:\windows\clouds.bmp). For available images, see [Images and Icons, p.153](#) in the *User's Guide*. Transparency indicators can also be applied to **GRID** bitmaps, where **T**=use upper left most pixel colour and **G**=use RGB:192,192,192; e.g., X'BITMAP\$="C:\pvxdev\resource\stop.bmp,t" X'BITMAP\$="!Stop,g"
- BitmapPosition** [BUTTON](#) [CHECK_BOX](#) [RADIO_BUTTON](#) [TRISTATE_BOX](#)
Bitmap position: **1**=Left of text; **2**=Right of text; **3**=Above text; **4**=Below text.
- BitmapPosition\$** [CHART](#)
Bitmap position/appearance of chart. Predefined positions include: TOPLEFT, LEFT, BOTTOMLEFT, TOPRIGHT, RIGHT, BOTTOMRIGHT. These preserve bitmap size and proportions. Use **STRETCH** parameter to force the bitmap to be stretched within the chart's window. The **TILE** paramter creates a "tile" effect multiplying the original bitmap to cover the entire chart's window. *Default:* "TOPLEFT".
 A custom position may be defined using syntax: "x:y:column:line". With this syntax, the bitmap is positioned within the given rectangle. Proportions and the size of the bitmap are altered to fit the rectangle.
- BottomBorder** [GRID](#)
Bottom border of cell (thickness): **0** to **3** pixels. *Default:* **0**.

BottomLeftTick\$	GRID <i>Bottom left tick.</i> When set to a colour, this displays a tick in the bottom left corner of the cell. Colour names are listed under Colour Properties, p.727 . <i>Default:</i> "DEFAULT".
Calendar\$	MULTI_LINE <i>Invoke Month Calendar Control.</i> For parameters, see Calendar, p.219 .
CascadeState	TREE_VIEW <i>Control cascading of states.</i> See State Indicators, p.731 .
CellFormat\$	GRID <i>Cell format mask.</i> <i>Default:</i> null. See FMT= option, MULTI_LINE Directive, p.215 .
CellHiLight	GRID <i>Cell selection highlight:</i> 0 =Cell not highlighted; 1 =Focus rectangle on cell; 2 =Cell highlighted (selected); 3 =Cell highlighted and has focus rectangle. <i>Default:</i> 1.
CellImpliedDecimal	GRID <i>Cell Implied Decimal:</i> Controls implied decimal input on cell by cell basis.
CellLeft	GRID <i>Cell Left Position:</i> Relative X position for cell.
CellTag\$	GRID <i>Maintain hidden tag string on cell.</i> This tag can hold internal user-defined information about the cell.
CellTbl\$	GRID <i>Translation table:</i> Returns table of multi-character values representing selections.
CellTblWidth	GRID <i>Translation table width:</i> Sets length of each item in the ' CellTbl ' property. Positive values only. Default is 1. Zero indicates that ' CellTbl\$ ' contains a delimited string, with the last character being the delimiter. Setting a translation table on column default (i.e., entire column) or on an individual cell in the column is <i>not supported</i> .
CellTip\$	GRID <i>Tip for Cell:</i> Tip message.
CellTop	GRID <i>Cell Top Position:</i> Relative Y position for cell.
CellType\$	GRID <i>Grid cell type.</i> For a list of available cell type values, see <i>Cell Types</i> under the GRID Directive, p.146 . <i>Default:</i> "Normal".

CellTypeList\$	GRID <i>List of supported cell types</i> (described above). Delimited list where last character is delimiter.
Children	TREE_VIEW <i>Number of direct descendant children for current item set by 'Item'.</i>
Col	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Screen position (column) of control.</i>
Cols	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Width of control in column units.</i>
Column	GRID LIST_VIEW <i>Column number of grid or list view.</i>
ColumnClicked	LIST_VIEW <i>Column Selected selected by the user.</i> Indicates if column is selected by the user. If a column was not clicked on, or the selected item changes by means other than the mouse then the property will return zero.
Column\$	GRID <i>Grid column name.</i>
ColumnNames\$	GRID <i>Comma-separated list of Grid column names.</i>
ColumnPixels	GRID <i>Width of column in pixels.</i>
ColumnSizeLock	GRID <i>Column resizing by user: 0=Off; 1=On.</i> Valid only when Resizable <> 0. <i>Default: = 0.</i>
ColumnsWide	GRID LIST_VIEW <i>Number of columns.</i> If a grid's dimensions have been changed using this property, the default settings must be re-set.
ColumnWidth	GRID LIST_VIEW <i>Width of column in column units.</i>
CurrentCellColour\$	GRID <i>Background Colour for current cell.</i> Valid colour names are listed under Colour Properties, p.727. <i>Default: "DEFAULT"</i> .
CurrentCellColor\$	<i>Background Colour for current cell – US spelling.</i> See above.
CurrentColumn	GRID <i>Current column with focus.</i>

CurrentItem	DROP_BOX LIST_BOX LIST_VIEW TREE_VIEW <i>Current item with focus/selected. Default: 1 (0, if no data) .</i>
CurrentPoint	CHART <i>Current data point. Default: 1 (0, if no data) .</i>
CurrentRow	GRID <i>Current row with focus. Default: 1.</i>
CurrentSet	CHART <i>Current dataset. Default: 1 (0, if no data).</i>
CtlName\$	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Control type.</i> This value can be one of the following: "MULTI_LINE" (see MULTI_LINE Directive, p.215) "LIST_BOX" (see LIST_BOX Directive, p.178) "LIST_VIEW" (see LIST_BOX Directive, p.189) "TREE_VIEW" (see LIST_BOX Directive, p.192) "DROP_BOX" (see DROP_BOX Directive, p.96) "VARLIST_BOX" (see VARLIST_BOX Directive, p.360) "VARDROP_BOX" (see VARDROP_BOX Directive, p.354) "BUTTON" (see BUTTON Directive, p.34) "CHECK_BOX" (see CHECK_BOX Directive, p.47) "TRISTATE_BOX" (see TRISTATE_BOX Directive, p.344) "RADIO_BUTTON" (see RADIO_BUTTON Directive, p.265) "V_SCROLLBAR" (see V_SCROLLBAR Directive, p.365) "H_SCROLLBAR" (see H_SCROLLBAR Directive, p.153) "GRID" (see GRID Directive, p.143) "CHART" (see CHART Directive, p.43).
DraggedColumn	GRID <i>Column number dragged from.</i> This indicates the column number (cell) where dragging started. <i>Default: 0</i> . See Drag and Drop, p.733 .
DraggedRow	GRID <i>Row number dragged from.</i> This indicates the row number (cell) where dragging started. <i>Default: 0</i> . See Drag and Drop, p.733 .
DroppedOn	DROP_BOX LIST_BOX LIST_VIEW TREE_VIEW VARDROP_BOX VARLIST_BOX <i>Index number in list box.</i> This indicates the index number that dragged items were dropped on to; if items are not dropped on to a specific line, 0 is returned.

- DroppedOnColumn** **GRID**
Column number where dropped. This indicates the column number (cell) where the mouse was released/items dropped. See [Drag and Drop, p.733](#).
- DroppedOnRow** **GRID**
Row number where dropped. This indicates the row number (cell) where the mouse is released/items dropped. See [Drag and Drop, p.733](#).
- Edit** **TREE_VIEW**
Control direct editing by user. This enables the automatic editing of item text; i.e., the user can double click on an item to change its value. (See OPT="E" under [Tree View, p.192](#)) **1**=allow editing, **0**=no editing, **-1**=force current item into edit mode. *Default: 0*.
- Enabled** **BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR**
Enabled indicator: 1=On, 0=Off. Default: 1.
- EnterMode** **GRID**
*Set movement for **Enter** key: 0=Normal Processing (edits cell); 1=Move right across a grid by column, exit on last column; 2=Move right across a grid by column, wrap to first column; 3=Move down by row, hold on last row; 4=Move down by row, return to column 1, hold on last row. Default: 0.*
 Using **Shift**, with these key modes reverses the direction. The **TabMode** value is automatically applied to **Enter** if the '+E' mnemonic has been set; however, **EnterMode** can be changed to a different value, if needed.
- Eom\$** **BUTTON CHART CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX**
Last change terminator. Refer to each control directive for specific eom\$ character value.
- ExcelStyle** **GRID**
Excel type grid: 0=No grid lines, 1=Lines. Default: 1.
- Expanded** **TREE_VIEW**
Node expanded: -2=Collapse selected level and all subordinates, -1/0=Collapsed, 1=Expanded, 2=Expand selected level and all sub-ordinates. Default: 0.
- FaceColourBack\$** **CHART**
Colour for background face. Valid colour names are listed under [Colour Properties, p.727](#). *Default: "DEFAULT"*.
- FaceColorBack\$** *Colour for background face – US spelling. See above.*

- FaceColourBottom\$** **CHART**
Colour for bottom face (3D chart). Valid colour names are listed under **Colour Properties, p.727**. *Default:* "DEFAULT"
- FaceColorBottom\$** *Colour for bottom face (3D chart)* – US spelling. See above.
- FaceColourLeft\$** **CHART**
Colour for left face (3D chart). Valid colour names are listed under **Colour Properties, p.727**. *Default:* "DEFAULT"
- FaceColorLeft\$** *Colour for left face (3D chart)* – US spelling. See above.
- FillColour\$** **GRID**
Fill colour. Defines the background colour for unused regions. Valid colour names are listed under **Colour Properties, p.727**. *Default:* "DEFAULT".
- FillColor\$** *Fill colour* – US spelling. See above.
- Fmt\$** **CHART GRID LIST_BOX LIST_VIEW MULTI_LINE TREE_VIEW**
Control format definition. For more information, refer to the **FMT=** option as described for use with a specific control directive; e.g., **CHART Directive, p.43**.
- Focus** **BUTTON CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX**
Focus indicator: 1=Control has focus. Default: 0.
- Font\$** **BUTTON CHART CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX**
Font for text element/cell. Indicates font name, size, and optional properties. Refer to the **'FONT' Mnemonic, p.609** for details. **CHART** supports **B, I, and U** options only.
- Footer\$** **CHART**
Chart footer.
- GenerateScrollEom** **MULTI_LINE**
Generate EOM value for scroll wheel actions. 0=Off, 1=On. Default: 0. This enables generation of EOM values (scroll up=0x26, down=0x28, left=0x25, right=0x27).
- Height** **BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR**
Height of control in pixels.

HoverColour\$	BUTTON CHECK_BOX LIST_BOX LIST_VIEW RADIO_BUTTON TRISTATE_BOX <i>Hover colour.</i> Highlights text when the mouse moves over its location. Valid colour names are listed under Colour Properties, p.727 . <i>Default:</i> "DEFAULT".
HoverColor\$	<i>Hover colour</i> – US spelling. See above.
hWnd	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Windows handle for control.</i>
Id	RADIO_BUTTON <i>Radio button index to reference.</i>
ImageCount	BUTTON CHECK_BOX RADIO_BUTTON TRISTATE_BOX <i>Number of images contained in a bitmap button.</i> 1 to 4. <i>Default:</i> 0.
ImpliedDecimal	GRID MULTI_LINE <i>Implied Decimal:</i> Controls implied decimal input in grids and multi-lines.
IndexMode	CHART <i>Set Index Mode.</i> This allows additional views of existing chart types to be opened; e.g., for a 2D column chart, setting IndexMode to 1 creates a clustered column chart. 1 =Natural number (1 .. <i>n</i>) indexing; 2 =Integer (0 .. <i>l</i>) indexing; 3 =Arbitrary <i>x</i> value indexing.
InsDelEnabled	GRID <i>Cell editing keys.</i> 0 =Off, 1 =On. <i>Default:</i> 0 . This enables use of Insert to begin cell editing and Delete for clearing the contents of a cell.
Item	DROP_BOX LIST_BOX LIST_VIEW TREE_VIEW VARDROP_BOX VARLIST_BOX <i>Index number of item in list.</i>
ItemCount	DROP_BOX LIST_BOX LIST_VIEW TREE_VIEW VARDROP_BOX VARLIST_BOX <i>Number of items in list.</i> See Load on Demand, p.729 .
ItemNeededCtl	LIST_BOX LIST_VIEW <i>Signal/CTL event number to generate when items are requested from the list box.</i> This property must be set prior to pre-declaring the number of items the list box is to have by setting the ItemCount property. See Load on Demand, p.729 .
ItemNeededFrom	LIST_BOX LIST_VIEW <i>Index number of the items requested.</i> The 'ItemNeededFrom and 'ItemNeededTo properties are set once the user scrolls the list box to request more items to be loaded. See Load on Demand, p.729 .

- ItemNeededTo** **LIST_BOX LIST_VIEW**
Index number of the items requested. See [Load on Demand](#), p.729.
- ItemState** **TREE_VIEW**
*Numeric value indicating the state of the item. 1-based, 0=No state indicator. There is a maximum of 15 states. This property is used in conjunction with **'StateBitmaps\$'**. See [State Indicators](#), p.731.*
- ItemTag\$** **TREE_VIEW**
*Maintain hidden tag string on item set by **'Item'**. This tag can hold internal user-defined information about the item such a file key, etc.*
- ItemText\$** **DROP_BOX LIST_BOX LIST_VIEW TREE_VIEW VARDROP_BOX VARLIST_BOX**
*Value of the current item set by **'Item'**.*
- JoinColumns** **GRID**
Merge two or more columns (left to right). Set this property in the column that starts the join (leftmost) to the total number of joined columns - that number of columns will be merged into one. For columns belonging to an existing join, this property returns a negative integer indicating the column's current position within the join.
- JoinRows** **GRID**
Merge two or more rows (downward). Set this property in the row that starts the join (uppermost) to the total number of joined rows - that number of rows will be merged into one. For rows belonging to an existing join, this property returns a negative integer indicating the row's current position within the join.
- Key\$** **BUTTON CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW
MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX
V_SCROLLBAR**
Hot key to jump to control.
- LabelLocation\$** **CHART**
Custom positioning of chart labels. Use syntax: "column:line" or "AUTOMATIC" (to set label locations to the default mode); e.g., x'LabelLocation\$="10:15". For more information on applying this property, see [Chart Label Reference](#), p.734.
- Left** **BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW
MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX
V_SCROLLBAR**
Left margin for control in pixels.
- LeftBorder** **GRID**
Left border of cell (thickness): 0 to 3 pixels. Default: 0.

LegendLocation\$	CHART <i>Location of chart legend.</i> Values include: TopLeft, Left, BottomLeft, TopRight, Right, and BottomRight.
LegendText\$	CHART <i>Legend text for a data set.</i>
Len	GRID MULTI_LINE VARDROP_BOX VARLIST_BOX <i>Input length of cell or line.</i>
Line	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Screen position of control.</i>
LineColour\$	TREE_VIEW <i>Colour of connecting lines in a tree view.</i> Valid colour names are listed under Colour Properties, p.727 . Default: "DEFAULT".
LineColor\$	<i>Colour of connecting lines in a tree view.</i> – US spelling. See above.
Lines	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Height of control in number of lines.</i>
LoadIOLIST\$	GRID <i>'LoadList\$ in compiled IOList format.</i> See Loading/Accessing by Row, p.733 .
LoadList\$	GRID <i>List of columns loaded.</i> Lists column names in the order they appear physically in the grid. Loading/Accessing by Row, p.733 .
Lock	GRID MULTI_LINE <i>Lock status.</i> 1=Lock, 0=Unlock. <i>Default: 0.</i>
LockColumns	GRID <i>Number of columns to lock into position, starting from column 1.</i>
LockRows	GRID <i>Number of rows to lock into position, starting from row 1.</i>
MarginBottom	CHART <i>Set line number for bottom chart margin.</i>

MarginLeft	CHART <i>Set column number for left chart margin.</i>
MarginRight	CHART <i>Set column number for right chart margin.</i>
Margins\$	CHART <i>Set all chart margins.</i> Use syntax: " <i>top:bottom:left:right</i> " or "AUTOMATIC" to set margins to the automatic (default) mode; e.g., X 'MARGINS\$="10:10:20:10" .
MarginTop	CHART <i>Set line number for top chart margin.</i>
MaxValue	H_SCROLLBAR V_SCROLLBAR <i>Maximum scroll bar value.</i>
MenuColumn	GRID <i>Column number on right-click.</i> This property indicates the column number of a selected cell on a right-click of the mouse. .
MenuCtl	BUTTON CHART CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX <i>CTL value on right-click.</i> This property reports/sets the CTL value to generate when an object is selected on a right-click of the mouse.
MenuRow	GRID <i>Row number on right-click.</i> This property indicates the row number of a selected cell on a right-click of the mouse.
MouseOver	LIST_BOX LIST_VIEW TREE_VIEW <i>Item number on mouse-over.</i> This property returns the item number of an object when the mouse pointer is over it. If the mouse is not over an object, -1 is returned. 0 is returned if the cursor is over an area in the control with no data.
Msg\$	BUTTON CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX <i>Message line text for the control.</i>
MultiSelect	GRID <i>Select multiple cells: 1=On; 0=Off. Default: 1.</i>
NotifyExpand	TREE_VIEW <i>Detect expand/collapse requests.</i> When set to non-zero value, this causes a tree view expand/collapse to generate an event with EOM code "+" (expand) or "-" (collapse).
Nul\$	GRID MULTI_LINE <i>Null input display text.</i> Replacement text/value in place of null; e.g., G 'NUL\$="asis".

NumPoints	CHART <i>Largest number of data points within data set.</i>
NumSets	CHART <i>Total number of data sets.</i>
OnFocusCols	MULTI_LINE <i>On focus, resize columns.</i> Returns to defined size when focus leaves. Assign 0 to reset.
OnFocusLines	MULTI_LINE <i>On focus, resize lines.</i> Returns to defined size when focus leaves. Assign 0 to reset.
OnFocusCtl	BUTTON CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX <i>On focus CTL event.</i> 0 is returned if no on focus CTL value is set up for the control.
OverlapEnabled	GRID <i>Allow cells to overlap: 1=Yes, 0=No. Default: 1.</i>
Parent	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Parent window handle.</i>
PointText\$	CHART <i>Point label values.</i> Single string of point label values where last character is delimiter.
PrefixData	TREE_VIEW <i>Control prefix on data loaded into tree view.</i> 0=No prefix on the data - data has bitmap option off; 1=Data loaded in the tree view can be prefixed with curly braces containing a bitmap, state value, and tag value (separated with semi-colons) - data has bitmap option on; 2=Returns same prefix with curly braces when data is read, and can be supplied when the data is written (as above).
Proportions2M	CHART <i>Proportions to margins.</i> 0=Off; 1=On. Sets the chart to the variable-proportion mode, which means that it is proportional to the current height-to-width ratio of the chart-window that contains the chart. <i>Default: 0.</i>
ProportionDW	CHART <i>Depth to width.</i> Sets percentage of chart-depth to chart-width for altering depth proportions of three-dimensional charts. Assigning a 0 zero forces default values according to chart type. Default is <code>ProportionDW=100</code> for pie chart, <code>ProportionDW=50</code> for other chart types.

ProportionHW	CHART <i>Height to width.</i> Sets percentage of chart-height to chart-width for altering proportions of two-dimensional charts. Assigning a 0 zero forces default values according to chart type. Default is <code>ProportionHW=5</code> for pie charts, <code>ProportionHW=100</code> for other chart types.
RangeMax	CHART <i>Set ceiling value of the Y-axis.</i> The chart view will be adjusted according to RangeMin and RangeMax values.
RangeMin	CHART <i>Set floor value of the Y-axis.</i> The chart view will be adjusted according to RangeMin and RangeMax values.
RangeText\$	CHART <i>Text of custom label for Stack Chart (one point per dataset).</i> Labels are placed on the right side. Labels must be added sequentially, starting from 1, up to the number of sets; e.g., <code>X'CURRENTSET=1, X'CURRENTPOINT=1, X'RANGETEXT\$="Label One"</code> .
Resizable	GRID <i>Enable grid resize by user:</i> 0 =Neither row or columns are resizable; 1 =Both columns and rows are resizable; 2 =Columns are resizable, rows are not; 3 =Rows are resizable, columns are not. <i>Default: 1.</i>
RightBorder	GRID <i>Right border of cell (thickness):</i> 0 to 3 pixels.
Row	GRID <i>Grid row reference.</i>
RowData\$	GRID <i>Row data based on.</i> 'LoadList\$. See Loading/Accessing by Row, p.733 .
RowHeight	GRID <i>Height of current row in lines.</i>
RowHiLight	GRID <i>Row selection highlight:</i> 1 =Row Highlight; 0 =Cell Highlight. This enables selection of a full row rather than limited to cells. Only locked cells are highlighted. One side effect is that when the cells are locked the grid becomes a fancy list view.
RowPixels	GRID <i>Height of row in pixels.</i>
RowsHigh	GRID <i>Number of rows defined in Grid.</i> If a grid's dimensions have been changed using this property, the default settings must be re-set.

- Scroll** **MULTI_LINE**
Set scroll bar types. 0=No scroll bars, 1=Vertical scroll bars, 2=Horizontal scroll bars, 3=Both scroll bars.
- ScrollWheel** **DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE TREE_VIEW V_SCROLLBAR VARDROP_BOX VARLIST_BOX**
*Set mouse scroll wheel support. 0=Scroll wheel support (all events go to parent window), 1=Scroll only if the control has focus (mouse can hover on or off control) 2=Scroll only if the control has focus (mouse *must* be hovered over control), 3=If control does not have focus, then scroll when mouse hovers over this control (otherwise, same as 1), 4=If control does not have focus, then scroll when mouse hovers over this control (otherwise, same as 2).*
- SelectColumn** **GRID**
Column number of selected cell. (Read Only) See [Multiple Selections, p.730](#).
- SelectCount** **GRID LIST_BOX LIST_VIEW TREE_VIEW**
Number of items/cells selected. Set this property to zero to de-select all items. See [Multiple Selections, p.730](#).
- SelectedChildren** **TREE_VIEW**
Number Of Child Items. Used in conjunction with '[SelectStateMask](#)' to return the number of child items with the desired state. (Children being entries on the tree that have no sub-ordinates). See [Multiple Selections, p.730](#).
- SelectIndex** **CHART GRID LIST_BOX LIST_VIEW TREE_VIEW**
Index to '[SelectItem](#)'. Set this property to point to a selected element; e.g., 1 points to the first item selected, 2 points to the second item selected, etc. See [Multiple Selections, p.730](#).
- SelectItem** **LIST_BOX LIST_VIEW TREE_VIEW**
Item number in list selected. This returns the sequential location within the list of the item being pointed to by '[SelectIndex](#)' property. It can also be used to select items; e.g., a bj 'selectitem=2 would select a list item whose index is 2. To deselect an item, use a minus sign; e.g, bj 'selectitem=-2. An obj 'selectitem=0 selects all list items if multi-select functionality (OPT=' # ') is being used.
- SelectLength** **MULTI_LINE VARDROP_BOX VARLIST_BOX**
Length of selected item. This reports the number of characters currently highlighted. This allows for cut, copy, and paste within a GUI input region.
- SelectOffset** **MULTI_LINE VARDROP_BOX VARLIST_BOX**
Position where highlight/cursor begins. This indicates where the highlight begins within the data or, if nothing is highlighted, the current cursor location. This allows for cut, copy, and paste within a GUI input region.

SelectRow	GRID <i>Row number of selected cell. (Read Only) See Multiple Selections, p.730.</i>
SelectStateMask	TREE_VIEW <i>State filter to apply. See State Indicators, p.731.</i>
SelectText\$	GRID MULTI_LINE VARDROP_BOX VARLIST_BOX <i>Text contained within the highlight. This allows for cut, copy, and paste within a GUI input region. See Multiple Selections, p.730.</i>
SelectValue\$	GRID <i>Value within selected field. Requires setting the SelectIndex property. See Multiple Selections, p.730.</i>
Sep\$	CHART DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE TREE_VIEW VARDROP_BOX VARLIST_BOX <i>Separator character between each field, column, or data point.</i>
SepLoad\$	CHART DROP_BOX GRID LIST_BOX LIST_VIEW TREE_VIEW VARDROP_BOX VARLIST_BOX <i>Separator character for each row or data set.</i>
SignalOnExit	DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE TREE_VIEW VARDROP_BOX VARLIST_BOX <i>Signal event on exit: 0=No signal on exit; 1=Signal on exit; 2=Grid only, signal when changing row or on exit (sets column value to 0 and row value to last row exited); 3=Grid only, signal when changing row or on exit (sets column and row values to zero).</i>
SignalOnly	BUTTON CHECK_BOX RADIO_BUTTON TRISTATE_BOX <i>Signal only-do not get focus: 0=Off; 1=On. This property can also be read, returning 1 or 0 to indicate if the control is to signal only and not get focus. Default: 0.</i>
SkipLockedCells	GRID <i>Skip over locked cells: 1=On; 0=Off. Default: 1.</i>
SmallJump	H_SCROLLBAR V_SCROLLBAR <i>Scroll bar small jump value.</i>
Sort	GRID LIST_VIEW TREE_VIEW <i>Column sorting. For <i>list view</i> and GRID controls, this sets the column number to sort by. (Locked rows are included in the sort.) Negative integers indicate descending order. In <i>tree views</i>, sorting values indicate the following: 1=Automatically sort data in ascending order; 0=Reset sort indicators; -1= cause current Item to be sorted; -2 cause the current Item and its descendants to be sorted. Default: 1.</i>

- Sort\$** **GRID**
Sort by Column Name. This sets a column name (inside quotes) to sort by. A minus sign indicates descending order; e.g., x'Sort\$ = "- col_name". (Locked rows are included in the sort.)
- SortCaseSensitive** **GRID**
Sort With Case Sensitivity. **1**=On; **0**=Off. *Default: 1.*
- SortColFmt\$** **GRID**
Sort by Date Format. Controls format for sorting columns based on date values rather than standard numeric/string data. *Year, Month and Day* code combinations of 1 to 3 characters. (e.g., Y, MY, DMY, MDY, YMD ...) Case insensitive. Use in conjunction with '**Column**' to determine where date sorting will be applied.
- SortOnHdrClick** **GRID LIST_VIEW**
Sort on Header Click. Controls how columns are sorted when the column header is clicked. For *grids*, use the '**Column**' property to set the column number ('**Column**' set to 0 affects entire grid). Property values are: **0**=Disable sorting when column header clicked (default); **1**=Enable sorting when column header clicked; **-1**=Use grid default setting (for single column mode only).
- For a *Report*-style list view control, property values enable/disable column sorting: **0**=Disable sorting when column header clicked (default); **1**=Enable sorting when column header clicked. If the list view was created with sorting disabled (OPT="q"), then the default is 0; otherwise, the default is 1.
- SortStyle** **GRID**
Sort without Nulls. Determines if cells with null values are to included in the sort. **Sortstyle** values indicate the following: **0**=Null values are sorted; **1**=Null values are excluded and cells will remain at the bottom of the grid (Excel-like style).
- StateBitmaps\$** **TREE_VIEW**
List of images used to display states. Separated by vertical bars; e.g., "!Stop|!Print". A maximum of 15 images can be applied to this property. See [State Indicators, p.731](#).
- SwapEnabled** **GRID**
Column swap enabled: **1**=Yes, **0**=No. *Default: 0.*
- TabMode** **GRID**
*Movement setting for **Tab** key:* **0**=Normal Processing (exits grid); **1**=Move right across a grid by column, exit on last column; **2**=Move right across a grid by column, wrap to first column; **3**=Move down by row, hold on last row; **4**=Move down by row, return to column 1, hold on last row. *Default: 0.*

Tbl\$	CHECK_BOX DROP_BOX LIST_BOX LIST_VIEW RADIO_BUTTON TRISTATE_BOX <i>Translation table.</i> Returns table of values representing selections. Single-character values are returned by default; however, multi-character translations are supported for DROP_BOX and LIST_BOX when length is set via the ' TblWidth ' property.
TblWidth	DROP_BOX LIST_BOX LIST_VIEW <i>Table width.</i> Sets length of each of the items in the ' Tbl\$ ' property. It can be set to any positive value (default is 1). Setting to zero indicates that ' Tbl\$ ' contains a delimited string, with the last character of the string being the delimiter character.
Text\$	BUTTON CHECK_BOX GRID RADIO_BUTTON TRISTATE_BOX <i>Text of item or label.</i> In a grid, this property applies to "Button" cell types only.
TextColour\$	BUTTON CHART CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX <i>Foreground text colour.</i> Valid colour names are listed under Colour Properties, p.727 . <i>Default:</i> "DEFAULT".
TextColor\$	<i>Foreground text colour – US spelling.</i> See above.
TickPerUnit	GRID <i>Display units in grid "ruler".</i> Sets the number of ruler-style "ticks" between numbers (units) displayed in the header cells (row and column) of the grid; e.g., ' TickPerUnit=8 ' will display a number every 8 ticks on the ruler and causes median points (at ' TickPerUnit/2 ') to appear slightly larger. Ruler numbers begin at zero and count upwards by whole numbers, but will be reset to zero again if the header cell contains a ' Value\$ ' greater than null. <i>Default:</i> 0.
TickPixels	GRID <i>Pixels between ticks in grid "ruler".</i> This sets the space in pixels between ruler-style "ticks" (marker lines) displayed in the header cells of the grid.
Tip\$	BUTTON CHART CHECK_BOX DROP_BOX GRID LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX <i>Tip message for control.</i>
Title1\$	CHART <i>Primary chart title.</i>
Title2\$	CHART <i>Secondary chart title.</i>
Top	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Top of control in pixels.</i>

TopBorder	GRID <i>Top border of cell (thickness): 0 to 3 pixels. Default: 0.</i>
TopLeftTick\$	GRID <i>Top left tick.</i> When set to a colour, this displays a tick in the top left corner of the cell. Valid colour names are listed under Colour Properties, p.727 . <i>Default: "DEFAULT".</i>
TrackColour\$	GRID <i>Cell tracking colour.</i> Header cells corresponding to a cell that currently has focus are switched to colour set by this property as the user moves around the grid. This provides a visual cue to the user for which column and row they are currently on. Only header cells that use their default colour will change to the tracking colour. Valid colour names are listed under Colour Properties, p.727 . <i>Default: "DEFAULT".</i>
TrackColor\$	<i>Cell tracking colour – US spelling.</i> See above.
Uppercase	GRID MULTI_LINE <i>Force text to uppercase. 1=On, 0=Off. Default: 0.</i>
Value	CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Current numeric item/value assigned.</i> For grid controls, <i>cell</i> value. For chart controls, <i>data</i> value (based on ' CurrentPoint and ' CurrentSet).
Value\$	CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Current string item/value.</i> For grid controls, <i>cell</i> value. For chart controls, <i>data</i> value (based on ' CurrentPoint and ' CurrentSet).
Visible	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Control visible flag: 1=Yes, 0=No. Default: 1.</i>
Width	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Width of control in pixels.</i>
XAxisLocation\$	CHART <i>X-Axis location.</i> ("Back" or "Bottom"). <i>Default: "Bottom"</i>
XAxisTitle\$	CHART <i>X-Axis title.</i>

YAxisLocation\$	CHART <i>Y-Axis location. ("Back" or "Left"). Default: "Left"</i>
YAxisTitle\$	CHART <i>Y-Axis title.</i>
ZAxisLocation\$	CHART <i>Z-Axis location. ("Bottom" or "Left"). Default: "Left"</i>
ZAxisTitle\$	CHART <i>Z-Axis title.</i>
_PropList\$	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Contains list of Properties. See Multi-Property Access, p.728.</i>
_PropSep\$	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Separator between values. See Multi-Property Access, p.728.</i>
_PropValues\$	BUTTON CHART CHECK_BOX DROP_BOX GRID H_SCROLLBAR LIST_BOX LIST_VIEW MULTI_LINE RADIO_BUTTON TREE_VIEW TRISTATE_BOX VARDROP_BOX VARLIST_BOX V_SCROLLBAR <i>Accesses property values. See Multi-Property Access, p.728.</i>

Colour Properties

The properties used to define colour attributes in various graphical control objects include the following:

BackColor\$	CurrentCellColour\$	FillColor\$	TextColour\$
BackHiligh1\$	FaceColourBack\$	HoverColour\$	TrackColour\$
BackHiligh2\$	FaceColourBottom\$	LineColour\$	
BottomLeftTick\$	FaceColourLeft\$	TopLeftTick\$	

To apply a colour to these properties, use one of the following valid colour names:

"DEFAULT"	"Dark Gray"
"Black"	"Dark Red"
"Light Red"	"Dark Green"
"Light Green"	"Dark Yellow"
"Light Yellow"	"Dark Blue"
"Light Blue"	"Dark Magenta"
"Light Magenta"	"Dark Cyan"
"Light Cyan"	"Light Gray"
"White"	

... or use a RGB code (RGB: *n n n* where *n*=0-255) or *user-defined* colour (refer to the '[COLOUR](#)' & '[_COLOUR](#)' Mnemonics, p.596).

Compound Properties

Initially, control object properties were introduced to provide dynamic access to the basic attributes of a control object in ProvideX. The original list of properties covered obvious features (**Height**, **Font\$**, **Text\$**) in graphical control objects, but as this list continues to expand with each new release of ProvideX, so does the variety of features that can be defined via properties.

Many of the latest properties are designed to provide characteristics and behaviour that go way beyond what can be defined by directives alone. Most attributes are defined using *individual* properties; however, some extended functionality is contingent on sets of two or more (*compound*) properties and require more complex manipulation. The following sections identify and describe these groupings.


Topics

- Multi-Property Access**, p.728
- Load on Demand**, p.729
- Multiple Selections**, p.730
- State Indicators**, p.731
- Drag and Drop**, p.733
- Loading/Accessing by Row**, p.733
- Grid Property Access**, p.734
- Chart Label Reference**, p.734

Multi-Property Access

The following properties allow an application to set/get values for more than one property for a control in a single command:

- '_PropList\$** Comma separated list of property names to read/write.
- '_PropValues\$** String that contains values for each of the properties in **'_PropList\$**.
- '_PropSep\$** Character used as a field separator between values.

The ability to handle multiple reads is useful for accessing properties across a WindX/JavX connection, particularly when dealing with an object that has a large number of properties such as a grid. This can boost performance and reduce the amount of network traffic.

To retrieve the value of multiple properties, first set **'_PropList\$**, then read **'_PropValues\$**; e.g.,

```
G1 '_PropList$="CurrentColumn,CurrentRow,Value$"
x$ = G1'_PropValues$
```

In this example, x\$ receives a string containing the values of CurrentColumn, CurrentRow, and Value\$, with each field separated by either the standard SEP field separator, or with the **'_PropSep\$** character.

Data can then be extracted using **READ DATA**; e.g.,

```
READ DATA FROM G1 '_PropValues$ TO IOL=MYIOL
MYIOL: IOLIST Col, Row, Value$
```

To set values, first set '**PropList\$** (if not already set), then set '**PropValues\$**; e.g.,

```
G1 '_PropValues$ = "1"+SEP+"2"+SEP+"Data"
or G1 '_PropValues$ = REC(IOL=MYIOL)
```

The advantage is that only one packet need be sent to WindX / JavX to either **SET** or **RETRIEVE** the values of multiple properties. *Do not* constantly read fields '**PropList\$** and '**PropSep\$** from the control in order to parse the data returned by '**PropValues\$**. This would defeat the purpose; e.g.,

```
READ DATA FROM G1 '_PropValues$, SEP=G1 '_PropSep$
TO IOL=CPL("IOLIST "+G1 '_PropList$)
```

The above example actually results in three exchanges with WindX: one to get '**PropSep\$**, the second for '**PropList\$**, and the third for '**PropValues\$**. The separator character and list should be maintained within the application code and should not be retrieved from the control.

Load on Demand (*List Box, List View*)

The following properties are used to improve list box load times:

'Itemcount	Defines number of items.
'ItemNeededCtl	CTL issued when data needed.
'ItemNeededFrom	Lowest item needed.
'ItemNeededTo	Highest item needed.

On-demand loading allows an application to load a list box with only those items the user actually scrolls into view. This reduces network traffic and file access since a list box is only loaded with those items required by the user. Also, it assures proper function of the scrollbar and its relationship to the list.

This feature requires the developer to pre-declare the number of items that the list box is to have (by setting the **'ItemCount** property). When the user scrolls items into view, the system generates a CTL event.

Upon receiving the CTL event (set by **'ItemNeededCtl**), the application queries **'ItemNeededFrom** and **'ItemNeededTo** to determine the index number and the number of items. The application then loads the list box with the contents of the specified items by setting **'Item** and **'ItemText\$**. If no elements are needed then **'ItemNeededFrom** and **'ItemNeededTo** will be zero. Once the value has been loaded into the **'ItemNeededTo**, ProvideX checks if further items are required and if so, it generates another CTL event.

In the case of a Report style list view, should the user request the list be sorted or attempt to auto-size the width of a column, the system will force a load of all list box elements before processing the request.

In some instances the contents of the list box may need to be shown prior to the contents being loaded, in which case the system will display 5 dots in place of the data.

For more information, refer to the [LIST_BOX Directive, p.178](#).

Multiple Selections (*Grid, List Box, List View, Tree View*)

The following properties are used to process multiple selections in *lists* and *grids*:

- 'SelectCount** Number of items/cells selected. (Set this property to zero to de-select all.)
- 'SelectIndex** Index to **'SelectItem**. Set this to point to a selected element; e.g., set to 1 to point at the first item selected, 2 to point at the second item selected, etc.

The following property applies to *lists* only:

- 'SelectItem** Item number in list selected. This returns the sequential location within the list of the item being pointed at by the **'SelectIndex** property.

Refer to the [LIST_BOX Directive, p.183](#) or the [GRID Directive, p.143](#) for more information on the controls described.

Example:

```
FOR I = 1 TO Lbl'SelectCount
  Lbl'SelectIndex = I; PRINT Lbl'SelectItem; NEXT
```

In Tree Views

In addition to the above properties, tree view controls support the following:

- 'SelectedChildren** Number Of Child Items.
- 'SelectedStateMask** State filter to apply.

Use **'SelectedChildren** in conjunction with **'SelectedStateMask** to return the number of child items with the desired state.

When **'SelectedStateMask** is set, the **'SelectCount**, **'SelectIndex**, and **'SelectItem** properties will reflect only those items that have the specified state; e.g., to find all items that have a state of one, set **'SelectedStateMask** to 1. **'SelectCount** will then indicate the number of items that have this state and sequencing through **'SelectIndex** and **'SelectItem** will return their item numbers.

In Grids

The following properties can be applied to *grids* for access to selected items based on the location defined by **'SelectIndex**:

- 'SelectRow** Row number of selected cell.
- 'SelectColumn** Column number of selected cell.
- 'SelectValue\$** Value within selected field.

'SelectText\$ Text contained within the highlight.

Example:

```
6000 ! 6000 - Example using multiple selection properties for a Grid
        control
6010 CELL_SELECTIONS:
6020 TOTAL_SELECTED=G1'SELECTCOUNT
6030 IF G1'ROWSHIGH<1 OR TOTAL_SELECTED=0 THEN EXIT
6040 FOR T=1 TO TOTAL_SELECTED
6050 LET G1'SELECTINDEX=T
6060 LET SL_VAL$=G1'SELECTVALUE$
6070 LET SL_COL=G1'SELECTCOLUMN
6080 LET SL_ROW=G1'SELECTROW
6090 LET G1'ROW=-1
6100 LET G1'COLUMN=SL_COL
6110 LET SL_COL_TITLE$=G1'VALUE$
6150 MSGBOX "Cell Value: "+PAD(SL_VAL$,50)+SEP+"Column:
        "+STR(SL_COL)+SEP+"Row: "+STR(SL_ROW)+SEP+"Column Title:
        "+PAD(SL_COL_T
6150:ITLE$,50),"Selection Item "+STR(T)+" of "+STR(TOTAL_SELECTED)
6160 NEXT T
6170 EXIT
```

State Indicators (*Tree Views*)

The following properties are used to create and process state indicators:

'ItemState State of **'Item**.
'StateBitmaps\$ List of images used to display states.
'AutoState Control auto toggling of state.
'CascadeState Control cascading of states.

State indicators are basically images that will appear in front of a list box entry that can be used to indicate whether the item has been selected or not. State indicators are currently supported for tree view list boxes and can set up during the definition of a list box control. For more information, refer to the [LIST_BOX Directive, p. 189](#).

Assigning Images

The application must set the **'StateBitmaps\$** property in order to define the number of images that will used in the display of state indicators. A maximum of 15 images can be assigned. All images must be of the same size/format and may specify transparency options. These images can be external or internal (see [Images and Icons, p. 153](#) in the *User's Guide*).

Toggling Between States

Once the bitmaps are set, each item/row/entry may set its **ItemState** property to determines what image is to appear next to the row text depending on the state. A maximum of 15 states can be assigned for each image. A state of 0 *zero* causes no state indicator to be displayed.

For example, assuming the list box is defined with 3 images. The first image will appear if the item state is 1, the second image will appear if the item state is 2 and the third image will appear if the item state is 3.

A CTL event will return `EOM="S"` if the property is set to a non-zero value. This is used to identify that the user clicked over the indicator state portion of the line, as opposed to elsewhere in the item. Applications that add state indicators to their existing logic should add a check for this EOM code.

Auto Toggling Of States

AutoState is a numeric property that controls auto toggling of states. If this property is set, state indicators can automatically be toggled without generation of a CTL event with `EOM="S"`.

The number of states that the system will toggle through is determined by the value set in this property or, if the property is set to 1, the number of bitmaps assigned to the tree view. In addition, when the user toggles a state indicator while holding down the **Shift**, all entries between the current entry and the last will be toggled to the new state of the current entry (in effect allowing for group select/deselect).

Cascading States

If the **CascadeState** property is set to non-zero, the system automatically cascades parent states to their children and correspondingly makes parent states representative of all of their children. Setting a parent state, either under program control or using the **AutoState** property in the tree view definition, will result in all subordinate children being set to the same state.

When a child state is set, its parent state will be set according to the state of all of the child's siblings; i.e., if all children are in a consistent state, the parent will be set to the same state. If a parent has children of various states (some on, some off), the parents state will be set to the value set in the **CascadeState** property.

For example, you could have three state indicators - *Off* (state 1), *On* (state 2), and *Partial* (state 3). You would set **AutoState** to 2 and **CascadeState** to 3 to have children that automatically toggle off/on and parents that will be *On* if all children are on, *Off* if all children are off, and *Partial* (state 3) if the children are not in a consistent state.

When cascading, only items with states will be affected. In addition, items without states will not affect their parents states, nor will changing the parent of an item without a state affect the children of that item.

Drag and Drop (*Grid*)

The following properties are associated with drag and drop functionality in grids:

'DraggedColumn	Column where the drag started from.
'DraggedRow	Row where the drag started from.
'DroppedOnColumn	Column dropped on..
'DroppedOnRow	Row dropped on.

When dragging off of a grid, the starting point must be a column or row header. If a column header is used and column swapping is enabled using the **'SwapEnabled** property, then the drop target must be a control other than the grid itself. This is because dragging a column header within the grid is how columns are swapped.

Example:

```

1600 ! 1600 - Drag and drop rows from g1 (grid 1) to g2 (grid 2)
1610 ROW_DROP:
1620 LET R=G1'DRAGGEDROW; IF R<1 THEN RETURN
1630 LET G1'ROW=R
1640 READ DATA FROM G1'ROWDATA$ TO IOL=G1'LOADIOLIST$
1650 GRID DELETE G1,0,R
1660 LET R=MAX(1,MIN(G2'DROPPEDONROW,G2'ROWSHIGH))
1670 GRID ADD G2,0,R; LET G2'ROW=R
1680 LET G2'ROWDATA$=REC(G2'LOADIOLIST$)
1690 RETURN

```

For more information, refer to the [GRID Directive, p.143](#).

Loading/Accessing by Row (*Grid*)

The following properties are used to load and access a grid by row:

'LoadList\$	Lists column names in the order they appear physically in the grid.
'LoadIOLIST\$\$	Same list of variables as 'LoadList\$, but in compiled IOList format.
'RowData\$	Returns or sets a complete row of data corresponding to the names defined in the 'LoadList\$. Each cell value is separated by the defined column separator character, 'Sep\$.

'LoadList\$ returns a list of column names in the order in which they physically appear within the grid object. When a grid is defined using **FMT=** in the **GRID** directive, or columns names are assigned to columns in a grid, **'LoadList\$** returns a list of those column names in their current display order. The compiled version, **'LoadIOLIST\$** simplifies the loading of grid rows from direct file contents or other IOList-based items; e.g.,

```

myGrid.ctl'RowData$=rec(myGrid.ctl'LoadIOLIST$)
GRID LOAD myGrid.ctl,1,row,rec(myGrid.ctl'LoadIOLIST$)

```

The **'RowData\$** property returns or sets a complete row of data in accordance to the names defined in **'LoadList\$** or **'LoadIOLIST\$**. The data returned is for the row currently identified by the **'Row** property, e.g.,

```
READ DATA FROM X'ROWDATA$ TO IOL=X'LoadIOLIST$
```

or

```
X'ROWDATA$=REC(X'LoadIOLIST$)
```

Because the properties **'LoadList\$**, **'LoadIOLIST\$** and **'RowData\$** use logical column names, column swapping has no impact.

For more information, refer to the [GRID Directive, p.143](#).

Grid Column-Row Reference

The following are used to identify specific cells in order to access a variety of grid properties:

'Row	Grid row reference.
'Column	Grid column reference number.
'Column\$	Grid column name.

To access an entire row, set **'Column** to 0. To access an entire column, set **'Row** to 0. The entire grid can be access by setting both **'Column** and **'Row** to 0. For more information, refer to the [GRID Directive, p.143](#).

Chart Label Reference

The following are used to identify specific labels (datasets and data points) in which to apply a variety of **CHART** properties:

'CurrentSet	Current data set.
'CurrentPoint	Current data point.

To reference legend labels for a specific dataset, set **'CurrentPoint** to 0. To reference a label for a specific point, set **'CurrentSet** to 0. For example,

```
C'CurrentSet=1,C'CurrentPoint=0,C'TextColour$="Light Magenta"
```

... changes the bar colour for dataset 1.

```
C'CurrentSet=0,C'CurrentPoint=2,C'Value$="ABC"
```

... changes the second point label.

'**CurrentSet** and '**CurrentPoint** combinations are also used to identify specific text elements (labels) on the chart to apply certain format properties. The value combinations for applying '**AutoScale**, '**Font\$**, '**TextColour\$**, '**LabelLocation\$**, are listed in the table below:

<i>Text Element</i>	<i>Set</i>	<i>Point</i>
<i>Title 1</i>	-1	-1
<i>Title 2</i>	-1	-2
<i>Footer</i>	-2	-1
<i>X-Axis Title</i>	-3	-1
<i>Y-Axis Title</i>	-3	-2
<i>Z-Axis Title</i>	-3	-3
<i>Legend</i>	1	0
<i>All Labels</i>	0	0
<i>X-Axis Text</i> (' TextColour\$ only)	0	-1
<i>Y-Axis Text</i> (' TextColour only)	0	-2
<i>Z-Axis Text</i> (' TextColour only)	0	-3

Special Files and Devices

Overview

ProvideX syntax also includes a set of special names that are used to access built-in virtual files, devices, and interfaces. These names are recognized and processed internally in the language at run time. The following ProvideX virtual files and output interfaces are covered in this chapter:

- *BITMAP*** Generates a 24-bit colour bitmap image in memory. This is for use in WindX or Windows only. See ***BITMAP* Virtual Bitmap, p.738**.
- *HTML*** Generates an HTML-formatted output file. See ***HTML* Print to HTML, p.740**.
- *MEMORY*** Establishes a memory-resident logical file. See ***MEMORY* Create & Use Memory File, p.741**.
- *PDF*** Generates a PDF-compatible output file. See ***PDF* PDF Print Interface, p.744**.
- *SYSTEM** ProvideX internal event handling. See ***SYSTEM Event handling Object, p.751**.
- *VIEWER*** Sends output to the ProvideX *Print Preview* facility. This is for use in WindX or Windows only. See ***VIEWER* Print Preview, p.752**.
- *WINDEV*** Sends hardcopy output to the Windows print subsystem in *raw* or *pass through* mode. This is for use in WindX or Windows only. See ***WINDEV* Raw Print Mode, p.756**.
- *WINPRT*** Sends hardcopy output to the Windows print subsystem using standard API access. This is for use in WindX or Windows only. See ***WINPRT* Windows Printing, p.760**.
- *XML** Interface for accessing, parsing and serializing XML documents based on the XML DOM (Document Object Model). See ***XML ProvideX XML Interface, p.764**.

BITMAP

Virtual Bitmap

Format `OPEN (chan)"*BITMAP*"[;options]`

Where:

chan Channel or logical file number; e.g., `OPEN (1)"*BITMAP*"`

options The following options are supported in the ***BITMAP*** syntax:

DPI=nnn Image resolution in dots per inch. The value given must be greater than 0. If not specified, the default is 120.

LENGTH=nnn.nn Image height in inches (to 2 decimal places). If not specified, the default is 11.

FORCE6X10=YES | NO Automatically adjusts the column width to 60% of the line height defined in font size specifications. This setting solves some minor alignment issues when printing proportional fonts to a graphical print device.

MARGINS=left:top:right:bottom Defines margins. Refer to the [Margin Settings](#) listed under ***PDF***.

ORIENTATION={LANDSCAPE | PORTRAIT} Swaps the width and length (rotates the image). If not specified, the default is **PORTRAIT**.

WIDTH=nnn.nn Image width in inches (to 2 decimal places). If not specified, the default is 8.5.

BITMAP Keyword, not case-sensitive. Special interface, enclosed in quotation marks within **OPEN** directive. (Include asterisks in syntax.)

Description ***BITMAP*** is a logical filename that can be used to capture graphical output (24-bit colour bitmap image) in memory. ***BITMAP*** can be opened and sent output, in much the same way as the ***WINPRT*** print interface. Once an image is generated in ***BITMAP***, it can then be sent to either the screen or printer via the **'PICTURE'** mnemonic. It could also be saved to a bitmap file (.bmp) via the **SAVE FILE** directive.



Warning: 24-bit colour at high DPI (resolution) can use very large amounts of memory. If there are insufficient operating system resources available, the result will be an Error #15: Operating system command failed.

See Also ['PICTURE' Mnemonic, p.631](#)
[SAVE FILE Directive, p.298](#)
[Logical Printers, User's Guide](#)

Examples To display a generated image using the **'PICTURE'** mnemonic, specify the channel number associated with the ***BITMAP*** file preceded by a '#'.

```
0010 BEGIN ; OPEN (12)"*bitmap*"; PRINT 'CS',
0020 PRINT (12)'RECTANGLE' (@X(1),@Y(1),@X(10),@Y(5)),
0030 PRINT (12)'TEXT' (@X(12),@Y(2), "Howdy"),
0050 PRINT 'PICTURE' (@X(40),@Y(0),@X(79),@Y(24), "#12",0),
```

The following version of this example includes syntax that allows it to be used in a client-server application.

```
0010 BEGIN ; OPEN (12)"[wdx]*bitmap*"; PRINT 'CS',
0011 CALL "[wdx]*windx.utl;get_num", "LFO",F
0020 PRINT (12)'RECTANGLE' (@X(1),@Y(1),@X(10),@Y(5)),
0030 PRINT (12)'TEXT' (@X(12),@Y(2), "Howdy"),
0050 PRINT 'PICTURE' (@X(40),@Y(0),@X(79),@Y(24), "#"+STR(F),0),
```

For more information, refer to the *ProvideX Client-Server Reference*.

HTML*Print to HTML*

Format	<p>OPEN (<i>chan</i>)"*HTML*[:options]"</p> <p><i>Where:</i></p> <p><i>chan</i> Channel or logical file number; e.g., <code>OPEN (1) "*html*"</code></p> <p><i>options</i> The following are supported in the "*HTML*" syntax or as OPT=string\$ (file open options):</p> <p>BACK=nnnnnn Background colour using standard HTML RGB colour representation.</p> <p>FONT=fontname Default fixed-pitched font (e.g., Courier).</p> <p>FILE=path Output filename. If omitted, prompt is issued at runtime.</p> <p>SHOW Causes the file to be passed to default browser (Windows only)</p> <p>TEXT=nnnnnn Text colour using standard HTML RGB colour representation.</p> <p>TITLE=name Document title. The default is the program name.</p> <p>*HTML* Keyword, not case-sensitive. Special interface, enclosed in quotation marks within OPEN directive. (Include asterisks in syntax.)</p>
Description	<p>*HTML* is a logical print file that can be used with ProvideX. It allows printed reports, using normal fixed fonts, to be formatted for use with an HTML viewer such as a browser.</p> <p>The system will prompt for an output filename to store the resulting HTML document.</p>
See Also	Logical Printers, User's Guide
Examples	<pre>OPEN (1,OPT="FILE=Sample.htm;SHOW;FONT=Courier New;TITLE=Sample ;BACK=FFFFFF;TEXT=000000") "*HTML*" OPEN (1)"*HTML*;FILE=Sample.htm;SHOW;FONT=Courier New;TITLE=Sample ;BACK=FFFFFF;TEXT=000000"</pre> <p>The above examples create an HTML file called <code>Sample.htm</code> with the title "Sample". Courier New font is used with a white background and black text. The SHOW option causes the default browser to be invoked to display the file.</p>

MEMORY*Create & Use Memory File*

- Format**
1. *Create Memory-Resident Queue of Records:* **OPEN** (*chan*[,*fileopt*])***MEMORY***
 2. *Create Multi-Key File:* **OPEN** (*chan*[,*fileopt*])***MEMORY*** [**KEYDEF**=*key_def*\$]"

Where:

- chan* Channel or logical file number.
- fileopt* **BSZ=num** File option for specifying record size (number of bytes). The default is 1,024 bytes unless it is overridden via **BSZ=** .
- key_def*\$ String expression defining the key. The key definition can be single- or multi-keyed and may contain up to 96 segments for a total of 16 keys. Key names and the various attributes supported on VLR/FLR files may also be included in the key definition.
- *MEMORY*** Keyword, not case-sensitive. Logical filename, enclosed in quotation marks within **OPEN** directive. (Include asterisks in syntax.)

Description ProvideX supports a memory-resident logical file called ***MEMORY***. This file may consist of a queue of records that can be accessed by index or by single key, or it may be a multi-key file accessed by key, depending on the format used to create it. Create the memory-queue file and assign the given logical file number (channel) via the **OPEN** directive.

Use **CLOSE** (*chan*) to delete a ***MEMORY*** file and return memory to the system.

Format 1: Create Memory-Resident Queue of Records

OPEN (*chan*[,*fileopt*])***MEMORY***

Use this format to create a memory-resident queue of records. ProvideX recognizes ***MEMORY*** at run time and deals with it internally. Once ***MEMORY*** is open, you can gain I/O access to memory-resident records in the same manner as you can to Direct files (by record index or by key) using the following I/O directives:

- READ** [Read Data from File, p.271](#),
- READ RECORD** [Read Record from File, p.275](#),
- WRITE** [Add/Update Data in File, p.383](#),
- WRITE RECORD** [Write Record, p.386](#),
- CLOSE** [Close File, p.56](#),
- MERGE** [Read/Append Lines from File, p.206](#),
- REMOVE** [Delete Record from File, p.281](#).

The first **WRITE** determines the file access method; i.e., the **IND=** option specifies Indexed file handling, and **KEY=** specifies Direct file handling (externally-keyed files). The maximum key size for ***MEMORY*** files is 8192 characters. Adding records to a ***MEMORY*** file by **IND()**, pushes all the records at that index down one, and inserts a new record. To modify a record in an index file, remove the old record and

then insert the new one. The functions [IND\(\)](#), [p.457](#), [KEF\(\)](#), [p.466](#), [KEL\(\)](#), [p.467](#), [KEN\(\)](#), [p.468](#), [KEP\(\)](#), [p.469](#), [KEY\(\)](#), [p.470](#), and [RNO\(\)](#), [p.514](#) can be used with memory-queue files.

Format 2: Create Memory-Resident Multi-Key File

OPEN (*chan* [, *fileopt*]) ***MEMORY*** [*KEYDEF=key_def\$*]"

Use this format to create a memory-resident multi-key file similar to regular Keyed files. I/O directives and functions supported by Keyed files are also supported by this type of memory file. Due to the way this format is processed, performance is generally better than using Format #1 with key access.

Files are limited to 2GB in size. The current size, or amount of memory being used can be determined by querying `NUM(FIN(chan , "FILELENGTH"))` or `DEC(00+MID(FIN(chan) , 1 , 4))`. Extended records are supported by indicating a record size in excess of 31,000 byte; e.g., `BSZ=32000` (the **BSZ=** value, if specified, must be a positive value).



Note: The ability to access a memory file in a mixed mode (by **KEY=** and/or **IND=**) is not supported.

See Also

[DIRECT Directive](#), [p.89](#)
[KEYED Directive](#), [p.166](#)

Examples

The following example illustrates how to open and use ***MEMORY*** using *Format 1*:

```
00010 mmf=hfn
00020 open (mmf) " *memory*"
00030 print 'CS'
00040 input "Name (Press F4 to end):",name$
00050 if ctl=4 then goto 0100
00060 input "Address:",addr$
00070 input "Position:",pos$
00080 write (mmf,key=name$)iol=mmfLst
00090 goto 0030
00100 print 'CS'
00110 select iol=mmfLst from mmf begin "" end "z"
00120 print "Name:",name$
00130 print "Address:",addr$
00140 print "Position:",pos$
00150 print "-----"
00160 next record
00170 end
00180 mmfLst: iolist name$,addr$,pos$
```

The following example illustrates how to open and use ***MEMORY*** using *Format 2*:

```
0010 mmf=HFN
0020 OPEN (mmf)"*memory*;keydef=[1:1:20],[2:1:60],[3:1:20]"
0030 PRINT 'CS'
0040 INPUT "Name (Press F4 to end):",name$
0050 IF CTL=4 THEN GOTO 0100
0060 INPUT "Address:",addr$
0070 INPUT "Position:",pos$
0080 WRITE (mmf)IOL=mmfLst
0090 GOTO 0030
0100 PRINT 'CS'
0110 SELECT IOL=mmfLst FROM mmf,KNO=1 BEGIN "" END "z"
0120 PRINT "Name:",name$
0130 PRINT "Address:",addr$
0140 PRINT "Position:",pos$
0150 PRINT "-----"
0160 NEXT RECORD
0170 END
0180 mmfLst: IOLIST name$,addr$,pos$
```

PDF**PDF Print Interface**

Formats

1. *Open PDF File*: **OPEN** (*chan[,fileopt]*)*PDF*[:*option*][:*option*] [...]"
2. *PDF via WindX*: **OPEN** [*INPUT*] (*chan[,fileopt]*)[*WDX*]*PDF*[:*option*][:*option*] [...]"

Where:

- chan* Channel or logical file number.
- fileopt* File options. Supported options for opening *PDF* include:
ERR=stmtref Error transfer.
OPT=option PDF output parameters (described below).
 To obtain the current **OPT=** value, use the **OPT() Function, p.495**.
- option* Supported parameters for defining the PDF output. (See ***PDF* Output Parameters** described below.)
- *PDF*** Keyword, not case-sensitive. Special device filename, enclosed in quotation marks within **OPEN** directive. (Include asterisks in syntax.)
- [*WDX*] File tag for directing output to a PDF on a WindX client machine instead of the server.

Description

Use ***PDF*** for directing output to a PDF (Postscript Document Format) compatible file. This interface operates in a similar manner to ***WINPRT***. If the output file name is omitted, a dialogue appears at run time for users to specify the path, PDF name and properties.

An **OPEN** may specify a number of semicolon separated options for the PDF. These may be included as part of the path, or within an **OPT=** clause:

OPEN(*chan*)*PDF* [:*option*] [:*option*] [...]"

OPEN(*chan*,**OPT=**"*option;option;...*")*PDF*"

***PDF* Output Parameters**

The following options can be used to define PDF output.

- ASIS** Uses last defined settings and does not present the user with a window to enter parameters. If the output file already exists, it will be overwritten.
- DISPLAY** Brings up an input window for the user to enter information regardless if it is needed.
- OVERWRITE** Indicates that overwriting the output file is OK, no need to verify with the user.
- FILE=name** Specifies the output (pdf) file pathname. If omitted, a dialogue appears at run time for users to specify the path, file name, and properties.

FORCE6X10=YES|NO Automatically adjusts the column width to 60% of the line height defined in font size specifications. This setting solves some minor alignment issues when printing proportional fonts to a graphical print device. Historically, most fixed-width fonts adhered to a 6x10 ratio of 6 lines to the inch and 10 characters per inch; i.e., for a character width that is 60% of the line height.

FORM=FormName Allows specification of forms names (see [Forms Handling](#)).

PAPERSIZE=FormNumber

Allows specification of form given internal form number (see [Forms Handling](#)).

ORIENTATION=LANDSCAPE | PORTRAIT

Defines how the report is intended to print.

MARGINS=left:top:right:bottom

Defines margins (See [Margin Settings](#)).

TITLE=string Adds a *Title* tag to the PDF Document.

SUBJECT=string Adds a *Subject* tag to the PDF Document.

AUTHOR=string Adds an *Author* tag to the PDF Document.

PRODUCER=string Adds a *Producer* tag to the PDF Document.

CREATOR=string Adds a *Creator* tag to the PDF Document.

KEYWORDS=string Adds a *Keywords* tag to the PDF Document.

Margin Settings

Values for margin settings are typically given in 1000^{ths} of an inch; however, the following also applies.

If the value contains the letter *I*, it is considered *inches*. If the value contains the letter *M*, it is considered *millimetres*. If the value contains a decimal point, or the value is less than 25, then it is not considered 1/1000th but as either inches or millimetres.

Example:

MARGINS="250:250:250:250" All 4 margins are 250/1000's (1/4 inch)

MARGINS="1:1:1:1" all margins are 1 inch

MARGINS="1i:1i:1i:1i" all margins are 1 inch

MARGINS="1.25:1.25:1.25:1.25" all margins are 1-1/4 inches

MARGINS="20m:20m:20m:20m" all margins are 20 millimeters.

General Information

If neither the **FORM=** or **PAPERSIZE=** is given, or the value for either of these two parameters cannot be found in the *Forms Library*, then the default **LETTER** size (8.5" x 11") size is assumed. If both **FORM=** and **PAPERSIZE=** are given, the **FORM=** option has priority.

For further details, see **Forms Handling**, p.748.

If the **ASIS** or **DISPLAY** options are not specified and no **FILE=** is given, then the system prompts the user for the file. If **ASIS** is specified, but no **FILE=** has been defined by a prior open to ***PDF***, then ProvideX will generate an Error #12: File does not exist (or already exists).

The filename specified by **FILE=** cannot have a **[WDX]** prefix. If the user wants output to go to a remote file they should issue the open as follows:

OPEN (nnn) "[WDX]*PDF*...".

In order to simplify the logic behind this process and allow for developer customization, whenever ***PDF*** is opened, the input pathname is forwarded for processing to the user-defined program ***ext/pdf** (if it exists) or the ProvideX-supplied program ***ext/system/pdf**. It determines if the selection window should be presented and validates the **FORM=** and **PAPERSIZE=** options.

These options passed may be retrieved via **PTH()** and **OPT()**.

Example:

```
OPEN (1) "*pdf*;FILE=/tmp/pvx.pdf; FORM=Letter:8.5in:11in"
```

Font Support

PDF supports the use of all TrueType and Type 1 fonts, in addition to the following basic PDF fonts:

```
"Courier", "Courier-Bold", "Courier-BoldOblique",
"Courier-Oblique", "Helvetica", "Helvetica-Bold",
"Helvetica-BoldOblique", "Helvetica-Oblique", "Times-Roman",
"Times-Bold", "Times-Italic", "Times-BoldItalic", "Symbol",
"ZapfDingbats"
```

To enable TrueType and Type 1 font support, ***PDF*** uses Libharu, an open-sourced library for generating PDF. **TCB(191)** will return 1 if ProvideX is compiled with LibHaru.

Supported Mnemonics and Functions

The following mnemonics are supported with *PDF*: 'TEXT', 'FONT', 'PEN', 'FILL', 'PICTURE', 'PIE', 'POLYGON', 'ARC', 'RECTANGLE', 'LINE', 'CIRCLE', 'OFFSET', 'LPI', 'CPI', 'MODE', 'Fn' and 'Bn', 'LF', 'CR', 'FF', 'CP', 'EP', 'SP', 'COLOUR' and 'COLOR, and named colours (e.g., 'WHITE').

The following language functions are supported: FIB(), FIN(), FID(), PTH(), OPT(), MXC(), MXL(), etc.

Creating Bookmarks

Bookmark *pointers* can be inserted into a generated PDF so that when the bookmark is selected in the document, it will display a specified page/line/column; e.g.,

```
PRINT (pdf_channel) 'OPTION' ( "BOOKMARK" , "bookmark_txt" ) ,
```

The *bookmark_txt* defines the text and hierarchy of the bookmark, with each level in the tree being separated by a character in the range of \$00\$ to \$1F\$ (less than a space). The bookmark tree is output in the order the bookmarks are issued. The maximum length for the complete bookmark text is 254 characters (including all levels and separators).

If a duplicate bookmark is issued, then the last occurrence will take precedence. This process enables the generation of a list of bookmarks whose order does not match that of the output document.

While bookmarks are rendered in the current text colour, they can be bolded by surrounding the 'OPTION' with 'BB'/EB' mnemonics, or italicized using 'BR'/ER' mnemonics. Both options can be specified together; e.g.,

```
PRINT (pdf_chan) 'BB' , 'OPTION' ( "BOOKMARK" ,
"Company$+$01$+Division$+$01$+Manager$+$01$" ) , 'EB' ,
```

When a PDF file is created with a bookmark, the system automatically sets the PDF file to display bookmarks upon opening. The location to which a bookmark refers/points is determined based on the last output printed to the PDF channel, i.e.,

- *Normal Text*, @(**c**), @(**c,l**), 'CR', 'LF', 'FF' - position on the page as of the end of printing the output.
- 'LINE', 'RECTANGLE', 'CIRCLE', 'TEXT', 'ARC' - starting point as defined by the mnemonic.

Text Mode Printing

PDF accepts character mode text:

```
PRINT (chan) "This is a sentence to output."
```

Limitations

The following are known limitations that control options under the ProvideX PDF interface:

- Invalid mnemonics are ignored.

- **'BO' & 'EO'** and user-defined mnemonics are ignored.
- Only **'PEN'** *solid*, *dashed*, or *no pen* style options are supported.
- Only **'FILL'** *solid* and *no fill* options are supported (no gradient fills, lines, or hashes).
- **'PICTURE'** is supported on the Windows platforms only (not on UNIX or Linux); however, any image format supported by the **'PICTURE'** mnemonic may be used (bmp, jpg etc).
- **'RECTANGLE'** *rounded* options are not supported.
- Compression is not supported for PDF files.
- *Output element skipping* is applied instead of *truncate*; i.e., PDF documents skip the processing of output elements that exist outside the page borders, or flow over a page border. ***WINPRT*** displays as much of the element as possible while truncating the portion that flows over a border.

Forms Handling

The ProvideX-supplied utility program ***ext/system/pdf** is embedded with a forms handling mechanism that validates and displays dialogues to the end user. To provide consistency with Windows forms handling, the PDF output paper size is determined via *form name* rather than by the physical dimensions of the paper.

For the purposes of the PDF handler, the **FORM=** option provides a form name followed by a colon, and include the paper size in terms of width and height. The paper size values are assumed to be in thousandths of an inch unless followed by the letter *I* (for *inches*) or the letter *M* (for *millimetres*).

The PDF processor within ProvideX will only utilize the width and height portions of the **FORM=** option. The form name is provided strictly as a reference for the display.

The *Form Library* is a keyed file. The PDF utility program will search for ***ext/forms.xx** then for ***ext/system/forms.xx**, where *xx* is a language code.

The properties for this file are:

Field

Form Number	Primary key, 3 digits. Explained below.
Name of Form	Alternate key, 24 characters
Width	
Height	
Units of Measure (I or M)	
Priority	Alternate key, Priority\$+ Form number

The ***ext/system/forms.en** file is pre-loaded with standard form sizes. The PDF selector window allows for additional forms to be added to the file. When the first form is added to this file by the end user, a copy of the file ***ext/system/forms.xx** is made in ***ext/forms.xx**, which is then used.

To simplify installation of applications with custom forms, when the selection window is presented and a **FORM=** is specified, the form name will be automatically added if it does not already exist in the *Forms Library*. This allows applications to be created with hard-coded **FORM=** clauses in their **OPEN** command—the system automatically creates the required form entry.

Form numbers are based on pre-defined form numbers used by MS Windows. As user-defined forms are added, new numbers will be assigned sequentially starting no lower than 256 (form numbers 0 to 255 are reserved).

Should the **OPEN** include a **PAPERSIZE=** option but no **FORM=** specification, the system will look up the form using the **PAPERSIZE=** value to determine what the proper form is. If there is neither a **FORM=** specification nor a valid **PAPERSIZE=** value, **LETTER** (8.5x11 inch) will be assumed. The **PAPERSIZE=** is ignored when a **FORM=** option is present.

The priority field is used to determine the load sequence for lists used to display the available forms in the system. By default, Letter, Legal, A4, A5, and a number of other standard form sizes are priority "0". All other forms have a priority of "5". User forms are added with a priority of "1" and appear near the top of the list when displayed—just below the standard form sizes.

Utility for Processing PDF Options

ProvideX includes PDF utility programs for processing options on an **OPEN**:

*ext/system/pdf	ProvideX-supplied program.
*ext/pdf	User-defined program.

Both utility programs have the following format:

ENTER *NewParameters\$, OldParameters\$, ERR=*next*

The utility will handle most PDF option string validation and forms processing internally. It ensures that the string that it gets passed to process has a valid **FORM=** option before it is returned to the actual PDF interface. It is basically a pre-processor for the PDF string.

Once the PDF string is accepted, the utility program returns it to ProvideX, which saves it as the default setting. In order to allow for application designers to access this default setting, the **OPEN** logic for ***PDF*** files allows the programmer to issue

OPEN INPUT (*chan*) **"*PDF*;ASIS"**

then query the **PTH()** and **OPT()** of the PDF file in order to obtain the settings. The **INPUT** clause prevents the system from erasing the output file. Optionally, to set default settings, the developer can issue:

OPEN INPUT (*chan*) **"*PDF*;new settings"**

PDF Message Selection Box (Graphical)

The user will be presented with a dialogue box to allow for selection of the output file, form, and orientation. When a file pathname has been given, the system checks to see if the file exists—if so, the user is asked if they wish to overwrite the output file.

The Form drop box contains the list of all the known form size in the system with the currently selected form name highlighted/selected (the default being Letter). The dialogue also has a New button for adding a new form description.

Pressing the **ACCEPT** button in the message box results in the form description being saved/added to the form library and the identified form being set as the output form type.

PDF Message Selection Box (Text Mode)

The user will be presented with a character-mode window. When a file pathname has been given, the system checks to see if the file exists—if so, the user is asked if they wish to overwrite the output file.

Once the file has been given, a drop list of the potential forms is displayed. The currently selected item is highlighted. After selecting the form, the user must select the orientation from a drop list with choices Portrait or Landscape.

Once all the selections are made, the user can select OK or Cancel to confirm the input.

Capturing Output to *WINPRT* as PDF Output

Output to *WINPRT* can be automatically intercepted for PDF using the 'AP' system parameter. When 'AP' is set, if the user selects Output To File and includes a filename ending in .PDF, ProvideX looks at the options selected during the *WINPRT* Printer Selection dialogue and processes the output through *PDF* instead of *WINPRT*.

'AP' may be set on in one of 3 ways:

1. Via `SET_PARAM 'AP'`.
2. In an INI file [Config] section as `AutoEnablePDF=1`.
3. From the System Menu on a dialogue (drop-down menu from clicking the Icon in a window title bar), as Use Internal PDF Driver.

By setting the `AutoEnablePDF= -1` in the [Config] section of your INI file, you can remove the entry from the dialogue system drop down menu. However, you are still be able to turn the parameter 'AP' on and off programmatically.

See Also

['AP' System Parameter, p.656](#)
['FONT' Mnemonic, p.609](#)
[*WINPRT* Windows Printing, p.760](#)
[\[WDX\] Tag, p.801](#)
[Logical Printers, User's Guide](#)

*SYSTEM

Event Handling Object

Formats *Initializes Event Handling Object:* **DEF OBJECT** *obj_id*,"*SYSTEM"

Where:

***SYSTEM** Keyword used in defining ProvideX Event Handling Object.

obj_id Numeric variable that will be used to save the object reference.

Description This object simplifies event handling in ProvideX. Use this object to turn event handling on and off and to isolate any syntax changes from your programs. To prevent an application from looping endlessly, a limit of 64 nested events has been imposed. The interface supports the following syntax options:

SETTIMER (*secs*)

Method Call. Fires a "timeout" event in ProvideX on intervals the number of seconds indicated by *secs*. The timer will continue to fire timeout events until a **SETTIMER(0)** is issued. The handling method will not receive any value when the event occurs.

(*Windows Only*).

SIGNALONDATA(*chan*, 0|1)

Method Call. Fires a "data available" event on the channel number given. A 1 indicates that the function is enabled; a 0 indicates that it is not. The channel number (*chan*) must refer to only certain kinds of files: console, TCP, serial connection, pipes and I/O redirection. The handling method will receive the channel number when the event occurs.

SIGNALONCLOSE(*chan*, 0|1)

Method Call. Fires a "file close" event for the channel number given. A 1 indicates that the function is enabled; a 0 indicates that it is not. The handling method will receive the channel number

SIGNALONOPEN(0|1)

Method Call. Fires a "file close" event for the channel number given. A 1 indicates that the function is enabled; a 0 indicates that it is not. The handling method will receive the channel number.

SIGNALLOADCLASS(0|1)

Method Call. Fires a "load class" event whenever a class definition needs to be loaded into the system either through a **NEW** or **LIKE** directive. The handling method will be passed the name of the class required.

Once the event is serviced the system will re-check to see if the desired class is now present if not (or no event service provided) the standard load of a ".pvc" file will occur.

VIEWER

Print Preview

Formats

1. *Open Viewer for Preview*: **OPEN** (*chan[,fileopt]*)*VIEWER*[:*option*][:*option*] [...]"
2. *Stand-Alone Viewer*: *path*\PVXWIN32.EXE *VIEWER/VIEWER [-XT=1 -ARG *filename DELETE*]"
3. *Via WindX*: **CALL** "*WindX.utl;SPAWN",*VIEWER/VIEWER [-XT=1 -ARG *filename DELETE*]"

Where:

<i>chan</i>	Channel or logical file number; e.g., <code>OPEN (1) "viewer"</code>
DELETE	Optional keyword to erase the file when the viewer closes.
<i>fileopt</i>	File options. Supported options include: ERR=stmtref Error transfer. OPT=option Output parameters (described below). To obtain current OPT= value, use the OPT() Function, p.495 .
<i>filename</i>	Name of the file that the print job is stored in or being spooled to. The file must be a serial file and <i>not</i> locked. See the SERIAL Directive, p.302 . To override the normal requirement that a serial file be locked, use 'LU' System Parameter, p.673 .
<i>option</i>	Supported parameters for defining the PDF output. (See *VIEWER* Output Parameters described below.)
*VIEWER/VIEWER	Program name for launching Viewer as stand-alone application.
VIEWER	Keyword, not case-sensitive. Special interface, enclosed in quotation marks within OPEN directive.
*WINDX.UTL;SPAWN	ProvideX <i>utility/function</i> . See also: Format 10: [WDX] and *WindX.utl, p.806 .



Note: For use in *WindX* or *Windows* only.

Description

The ***VIEWER*** (*graphical print preview*) is a special device file for previewing reports exactly as they would appear when output via ***WINPRT***. This invokes the Viewer user interface, which is able to display currently-spooling print jobs, copy to the clipboard, and print jobs stored on the disk. Other features of this interface include:

- Handling of reports from 0 to 50,000 pages in size.
- View options 1 page, 2 pages, 2-page book style, and 4 pages at a time.
- Zooms from 10% to 400% in increments or custom levels with *fit-to-width* and *fit-to-window* zoom options,
- Portrait and landscape modes.

The different methods for starting and using the Viewer user interface are described below.

Format 1: Open Viewer for Print Preview

```
OPEN(chan)*VIEWER* [;option] [;option] [...]"
```

An **OPEN** may specify a number of semicolon separated options for invoking the Viewer interface. These may be included as part of the path, or within the **OPT=** clause; e.g., **OPEN(*chan*,OPT="*option*;*option*;...")*VIEWER***.

Two special options (**INLINE** and **ONCLOSE**) can be used to control how the preview is created at run-time. These and other options are described below.

VIEWER Output Parameters

The following options can be used to define PDF output.

INLINE	Does not invoke a new instance of ProvideX to run the Viewer, but uses the current instance (implies ONCLOSE). The Viewer takes control of the ProvideX session currently running, just as a called program would. This option enables the Viewer interface to be used as a <i>top window</i> that must be closed before the user can continue. Also, if you wish to maintain the current communications channel, you will be able to run the Viewer without spawning a new session; e.g., via WindX.
ONCLOSE	Invokes a new instance of the Viewer interface when the printer channel is closed.
TITLE=string	Add <i>Title</i> tag to preview. Can also be used to complete the information in the Windows Print Job window.
PRINTER=string	Associated printer name.
PAPERSIZE= num	Define paper size.
ORIENTATION=LANDSCAPE PORTRAIT	Swap output width for length, and vice versa.
SOURCE= num	Set specific paper source.
QUALITY= num	Specifies DPI, 0 (or <i>unspecified</i>) = printer default (<i>printer output only</i>).
COLLATE= YES NO	Collate paper (<i>printer output only</i>).
COLOUR= YES NO	Coloured print (<i>printer output only</i>).
COPIES= num	Set number of copies (<i>printer output only</i>).
RANGE= num from:to	Select only specific page number, or range to print.
MARGINS= left:top:right:bottom	Define margins in 1/1000ths of an inch
COLUMNS= num	Minimum number of columns (<i>text based reports only</i>).

Rows= num	Minimum number of rows (<i>text based reports only</i>).
PAGINATIONAT= num	Auto form feed at row.
SCALETOFIT	Resize report to fit paper size (<i>text based reports only</i>).
FONT=fontspec	Default font for reports. Defaults to "Courier New, -10".
FONTSIZE=num	Default font size for reports.
SUPPRESSFIRSTBLANKPAGE	
SUPPRESSALLBLANKPAGES	
WATERMARKTEXT= string	
WATERMARKTEXTLOCATION= num	0=centered, 1=tiled, 2=top left, 3=top right, 4=bottom left, 5=bottom right
WATERMARKTEXTFONT= fontspec	
WATERMARKTEXTROTATION= num	Text rotation, angle in degrees.
WATERMARKIMAGE= string	Image name, assumes 100 pixels to the inch.
WATERMARKIMAGELOCATION= num	0=centered, 1=tiled, 2=top left, 3=top right, 4=bottom left, 5=bottom right

The following 6 items are used only if '**SP**', '**CP**', '**EP**' mnemonics are encountered in the report. If not set, then the viewer will attempt to set correct values for each. It is not necessary to set all of these items. Simply setting **SPCols** is enough. The values for '**CP**' / '**EP**' are calculated and the **Rows=** will all default to the same value.

SPCols=num	0 to 255, number of columns when in ' SP '.
SPRows=num	0 to 255, number of rows when in ' SP '.
CPCols=num	0 to 255, number of columns when in ' CP '.
CPRows=num	0 to 255, number of rows when in ' CP '.
EPCols=num	0 to 255, number of columns when in ' EP '.
EPRows=num	0 to 255, number of rows when in ' EP '.

Format 2: Stand-Alone Viewer Application

```
path\PVXWIN32.EXE "*"VIEWER/VIEWER -XT=1 -ARG filename[ DELETE]"
```

This format launches the Viewer as a standalone application (*with or without* opening a report file for previewing).

See Also **Logical Printers, User's Guide.**

Examples

The following example illustrates how to open and use the ***VIEWER*** for print preview:

```
0010 LET CHAN=UNT; OPEN (CHAN)"*VIEWER*"
0020 PRINT (CHAN)'FONT'("Courier New",-10),'DF',
0030 PRINT (CHAN)'FONT'("Arial",2),'TEXT'(@X(2),@Y(2),"Fonted Text"),
0040 PRINT (CHAN)'PICTURE'(@X(3),@Y(5),@X(43),@Y(30),"*win/nomads2"),
```



Note: Set the **BM** Mnemonic, [p.591](#), to **ON** to have the ***VIEWER*** send all data directly to the print file without interpretation. This allows you to send print jobs to any Windows printer that is available on the client PC. The ***VIEWER*** filters out the first and/or last page if either is blank.

The ***VIEWER*** does not return values for maximum column / line; i.e., in **MXC()** and **MXL()** functions. The following example illustrates how to approximate the number of columns:

```
CHAN=UNT; OPEN (CHAN)"*WINPRT*"
POINTS=-10
LOOP:
PRINT (CHAN)'FONT'("Courier New",POINTS),'DF',
IF MXC(CHAN)<132 THEN POINTS+=2; GOTO LOOP
LINES=MXL(CHAN),COLS=MXC(CHAN)
PRINT (CHAN)'AB', ! <--- aborts print job
CLOSE (CHAN)
OPEN (CHAN)"*VIEWER*"
PRINT 'FONT'("Courier New",POINTS),'DF',mm/line
```

WINDEV**Raw Print Mode**

Formats

1. *Open Device File*: **OPEN** (*chan[,fileopt]*)*WINDEV*[:*Q_name*]"
2. *Open for Read-Only Mode*: **OPEN INPUT** (*chan[,fileopt]*)*WINDEV*[:*Q_name*]"
3. *Open for [WDX]*: **OPEN [INPUT]** (*chan[,fileopt]*)[WDX]*WINDEV*[:*Q_name*]"

Where:

- chan* Channel or logical file number; e.g., `OPEN (1) " *windev* "`
- fileopt* File options. Supported options for opening *WINDEV* include:
ERR=stmtref Error transfer.
OPT=string\$ File open options. See [File OPEN Options, p.233](#).
 To obtain the current **OPT=** value, use the [OPT\(\) Function, p.495](#).
- Q_name* Print queue to open. If this information is omitted, the standard Windows printer dialogue appears at run time (users can select from a list of printers and properties.) Valid *Q_name* options include:
- OS name of an existing physical print queue on the local Windows machine; e.g., `open (14) " *WINDEV* ;HP LaserJet "`.
 - OS name of a print queue **ON** actual resource (via UNC path `\\machine\resource` for shared printers, or `LPT#` for direct access to a local port); e.g.,
`OPEN (14) " *WINDEV* ;HP LaserJet ON \\Main_Server\HP Laser "`
`OPEN (14) " *WINDEV* ;LP ON LPT1 "`
 - One of the following queue selection keywords:
 - ASIS** Most recently selected printer and properties.
 - DEFAULT** Printer currently "Set As Default" in the system; e.g.,
`OPEN (30 ,ERR=9900) " *WINDEV* ;DEFAULT "`.
 - NORMAL** Normal dialogue with page range (no paper size, source tray).
 - SETUP** Setup dialogue with paper size, source tray (no page range).

The *Q_name* is optional. *WINDEV* does not support queue option settings (graphical printer properties in the Windows Printer dialogue box such as: `orientation=landscape;copies=3`). See also **Printing in Windows, User's Guide**.

- [WDX] *WINDEV* is specific to Windows operating systems. Under UNIX, ProvideX automatically directs *WINDEV* access to the WindX client; e.g., `OPEN (14) " *windev* " !` For the PC client from UNIX host.
- In a Windows Server environment, the *WINDEV* printer is opened relative to the host unless you prefix the printer name with [WDX]; e.g.,
`OPEN (14) " *winprt* " !` For Windows Server
`OPEN (14) "[WDX]*winprt* " !` For client workstation

- *WINDEV* Keyword, not case-sensitive. Special device filename, enclosed in quotation marks within **OPEN** directive. (Include asterisks in syntax.)

Description Use ***WINDEV*** with your **OPEN** and/or **OPEN INPUT** directives to gain access to the Windows print subsystem in *raw* or *pass-through* mode. For standard API access, use ***WINPRT*** [Windows Printing, p.760](#).



Note: For use in *WindX* or *Windows* only.

You can identify the LPT for direct local access to the port or use UNC's (Universal Naming Conventions) for transmissions to a shared resource. For both LPT and UNC use, note that you can use raw escape sequences but graphical printing is not supported. LPT identification is not recommended under Windows Server.

Normally, you can take advantage of ***WINDEV*** to send data to the printer without having the driver strip out escape codes (for instance, to pass PCL code to the printer). That is, you *can* use escape sequences with ***WINDEV***, but these must be both valid and supported by your given printer and print driver.

ProvideX recognizes ***WINDEV*** as a special device file in your **OPEN [INPUT]** directive and deals with it internally in the language at run time. ProvideX returns an Error #12: File does not exist (or already exists) on the **OPEN** if no printers are installed or if the user presses the Cancel button in a printer selection dialogue. (This error can also occur if no printer is "Set As Default".)

Raw Printing Behaviour

ProvideX supports *raw* mode to send raw data to a Windows printer. You can control raw printing mode using the '**RP**' [System Parameter, p.684](#). This parameter's default is **ON**. When you turn it **OFF**, ProvideX uses the old *pass through* mode.

The printer drivers shouldn't (but might) strip escape sequences from the data you send to ***WINDEV***. Support for raw mode is printer and driver-dependent. Some drivers can destroy certain escape sequences (removing them from the data stream). If escape sequences disappear from your ***WINDEV*** print jobs, check the sequences for validity and check printer/driver appetites with their manufacturers.

Format 1: Open Device File

OPEN (*chan[,fileopt]*)*WINDEV*[:*Q_name*]"

Use this format to open the ***WINDEV*** device file to pass print jobs through in raw mode to the given queue on your open channel. See [Printing in Windows, User's Guide](#).



Note: Some device drivers issue an extra blank page (some even hang) if an open channel is closed too quickly or if nothing is printed to the channel before it's closed. Use the **OPEN INPUT** directive (described next) to bypass problems of this nature.

Format 2: Open for Read-Only Mode

OPEN INPUT (*chan[,fileopt]*)*WINDEV*[:*Q_name*]"

Use the **OPEN INPUT** directive to open *WINDEV* in read-only mode when you only want to determine the name and properties of a printer without sending a physical job. With an **OPEN INPUT** directive, you can open the printer, query the printer's properties, and close the channel without starting a physical job.

Example:

Use the **WINPRT_SETUP READ PROPERTIES** directive or **MXC()** and **MXL()** functions without generating a FormFeed:

```
IF WDX%<>$00$ THEN OPEN INPUT (30)"*WINDEV*;ASIS"
C=MXC(30)+1 ! Zero-based. For this printer MXC(30)=79, C=80 (0-79 columns)
L=MXL(30)+1 ! Also zero-based.
WINPRT_SETUP READ PROPERTIES WHAT_PROP$
! WHAT_PROP$ returns printer-specific list (items such as COPIES=1, OFFSET=0:0)
```



Note: The **'FONT'** mnemonic does not work with text-mode printers. To control fonts on a text mode device, send raw escape sequences to the printer using *WINDEV*, UNC (Universal Naming Conventions), or direct access to LPT ports. Your choice of fonts via *WINDEV*, UNC, or LPT is limited to the fonts supported by the given printer.

Format 3: Open for [WDX]

OPEN [INPUT] (*chan[,fileopt]*)[WDX]*WINDEV*[:*Q_name*]"

On a Windows Server, if you include [WDX] in your **OPEN [INPUT]** directive (e.g., **OPEN (30) "[WDX]*WINDEV*"**), then that signals ProvideX to direct any print jobs and dialogues to the WindX client PC, which will in turn use its Windows print subsystem to send jobs to the given printer.

If you are using *WINDEV* on an Windows Server and do not use [WDX] in your **OPEN** directive, then the printer selection dialogue will appear on the server console, and any print queue you name directly must exist on the Windows Server in the Control Panel printers folder.



Reminder: You must install and use WindX to use *WINDEV* in a UNIX environment; however, you do not need the [WDX] tag on a UNIX server because ProvideX automatically directs *WINDEV* access to your WindX client PC.

[WDX]*WINDEV* Escape Sequences

If you are opening [WDX]*WINDEV* and defining/creating mnemonics that will send escape sequences to the printer channel, you must send the mnemonic definitions to the WindX client instead of defining them on the server.

This is because internally there are actually two channels open: one ProvideX is using to route printing to WindX and one WindX opens, connected to the actual port on the client. Your mnemonic is run locally on the server's channel and is not sent to WindX for the remote client.

To send the mnemonic to the client in older versions of ProvideX/WindX, you can either:

- Create a device driver containing mnemonic definitions on the WindX client PC and then use a **CALL** directive; e.g., `CALL [WDX]*dev/your_driver_name , or`
- Use an **EXECUTE** directive from the server side for mnemonic definitions on the client; e.g., `EXECUTE "[WDX]MNEMONIC (LFO)'XX'=. . . ."`.

For further information, refer to the [MNEMONIC Directive, p.210](#) and the [\[WDX\] Tag, p.801](#).

See Also

[WINPRT_SETUP Directive, p.376](#)
[MXC\(\) / MXL\(\) Functions, p.488](#)
[Printing, User's Guide](#)
[\[WDX\] Tag, p.801](#)



WINPRT

Windows Printing

Formats

1. *Open Device File*: `OPEN (chan[,fileopt])*WINPRT*[:Q_name[Q_options]]"`
2. *Open for Read-Only*: `OPEN INPUT (chan[,fileopt])*WINPRT*[:Q_name[Q_options]]"`
3. *Open for [WDX]*: `OPEN [INPUT] (chan[,fileopt])[WDX]*WINPRT*[:Q_name[Q_options]]"`

Where:

- chan** Channel or logical file number; e.g., `OPEN (1) " *winprt* "`
- fileopt** File options. Supported options for opening *WINPRT* include:
ERR=stmtref Error transfer.
OPT=string\$ File open options. See [File OPEN Options, p.233](#).
 To obtain the current **OPT=** value, use the [OPT\(\) Function, p.495](#).
- Q_name** Print queue to open. If this information is omitted, the standard Windows printer dialogue appears at run time (users can select from a list of printers and properties.) Valid **Q_name** options include:
- OS name of an existing physical print queue on the local Windows machine; e.g., `open (14) " *WINPRT* ;HP LaserJet "`.
 - OS name of a print queue **ON** actual resource (via UNC path `\machine\resource` for shared printers, or `LPT#` for direct access to a local port); e.g.,
`OPEN (14) " *WINPRT* ;HP LaserJet ON \\Main_Server\HP Laser "`
`OPEN (14) " *WINPRT* ;LP ON LPT1 "`
 - One of the following queue selection keywords:
 - ASIS** Most recently selected printer and properties.
 - DEFAULT** Printer currently "Set As Default" in the system; e.g.,
`OPEN (30 ,ERR=9900) " *WINPRT* ;DEFAULT "`.
 - NORMAL** Normal dialogue with page range (no paper size, source tray).
 - SETUP** Setup dialogue with paper size, source tray (no page range).
- Q_name** may be optional, but it is required in order to override queue properties (see *Q_options*, below). See also [Printing in Windows, User's Guide](#).
- Q_options** Override printer properties. String expressions. You can override printer- and driver-specific values by assigning a new value to the queue (e.g., `copies=2` instead of `copies=1`). Use semicolons to separate items if you have a list.
- Q_options** must be preceded by a **Q_name**; i.e., `HP LaserJet` in the example below, or a queue keyword like **DEFAULT**).
- Specific properties are listed under [WINPRT_SETUP Properties, p.376](#). See also [Printing in Windows, User's Guide](#).

Example:

```
OPEN(30) " *WINPRT* ;HP LaserJet ;orientation=landscape ;copies=3 "
```


- [WDX]** *WINPRT* is specific to Windows operating systems. Under UNIX, ProvideX automatically directs *WINPRT* access to the WindX client; e.g.,
`OPEN (14) "*winprt*" !For the PC client from UNIX host.`
 In a Windows Server environment, you are opening the *WINPRT* printer relative to the host unless you prefix the printer name with the **[WDX]** tag; e.g.,
`OPEN (14) "*winprt*" !For Windows Server`
`OPEN (14) "[WDX]*winprt*" !For client workstation`
- *WINPRT*** Keyword, not case-sensitive. Special device filename, enclosed in quotation marks within **OPEN** directive. (Include asterisks in syntax.)

Description Use *WINPRT* with your **OPEN** and/or **OPEN INPUT** directives to gain standard API access to the Windows print subsystem. For raw or *pass-through* mode, use ***WINDEV* Raw Print Mode, p.756**.



Note: For use in *WindX* or *Windows* only.

The device driver for your given printer interprets the data you send to the *WINPRT* channel, then sends the output to the Windows spooling subsystem for transmission to its destination. You can identify the LPT for direct local access to the port or use UNC's (Universal Naming Conventions) for transmissions to a shared resource; however, LPT identification is not recommended under Windows Server.

ProvideX recognizes *WINPRT* as a special device file in your **OPEN [INPUT]** directive and deals with it internally in the language at run time. ProvideX returns an Error #12: File does not exist (or already exists) on the **OPEN** if no printers are installed or if the user presses the Cancel button in a printer selection dialog. (This error can also occur if no printer is "Set As Default".)

Some printer device drivers are unable to handle invalid *Q_options*; i.e., unknown property assignments and/or syntax errors (like `range=1, 5` instead of the correct `range=1:5`). The result can be unpredictable. The driver can even cause your ProvideX session to hang during the open. If you encounter unexpected problems, invalid *Q_options* for the given driver are the likely cause.



Note: Escape sequences are *not* allowed with *WINPRT* and may have an unpredictable effect on the device and/or printer driver. If you need access to the print subsystem to send PCL, escape sequences, etc., use ***WINDEV* Raw Print Mode, p.756**.

Format 1: Open Device File

OPEN (*chan[,fileopt]*)*WINPRT*[:*Q_name*[*Q_options*]]"

Use this format to open the *WINPRT* device file. ProvideX will recognize and deal with this special device file at run time to give you access to the Windows print subsystem. Then you can send print jobs to the given queue on your open channel. See **Printing in Windows, User's Guide**.



Note: Some device drivers issue an extra blank page (some even hang) if an open channel is closed too quickly or if nothing is printed to the channel before it's closed. Use the **OPEN INPUT** directive (described next) to bypass problems of this nature.

Format 2: Open for Read-Only Mode

OPEN INPUT (*chan[,fileopt]*)*WINPRT*[:*Q_name*[*Q_options*]]"

Use the **OPEN INPUT** directive to open *WINPRT* in read-only mode when you only want to determine the properties (*Q_options*) of a printer without sending a physical job. With an **OPEN INPUT** directive, you can open the printer, process your queries, and close the channel without starting a physical job.

Example:

Use the **WINPRT_SETUP READ PROPERTIES** directive, a **'FONT'(LIST)** graphics mnemonic or the **MXC()** and **MXL()** functions without generating a FormFeed:

```
IF WDX%<>$00$ THEN OPEN INPUT (30)"*WINPRT*;ASIS"
X$='FONT'(LIST*,30) ! Get font list
-:?X$
```

System,Fixedsys,Terminal,MS Serif,MS Sans Serif,Courier,Symbol,Small Fonts, Modern,FrameMakerSmallFont,Marlett,Arial,Courier New,Times New Roman, ... etc.



Note: The **'FONT'** mnemonic works with *WINPRT*, but not with any of the text-mode printers.

Maximum column and line values are zero-based. In the following example, the **MXC()** value returned is 79, for 0-79 = 80 columns:

```
C=MXC(30)+1 ! For this printer MXC(30) returns 79, C=80 (0-79)
L=MXL(30)+1 ! For this printer MXL(30) returns 55, L=56 (0-55)
```

Your string variable in reading properties returns a printer-specific list (here, for the **ASIS** printer):

```
WINPRT_SETUP READ PROPERTIES WHAT_PROP$
-:?what_prop$
RANGE=ALL;COLLATE=NO;COPIES=1;ORIENTATION=PORTRAIT;PAPERSIZE=1;SOURCE=1;R
ESOLUTION=300;OFFSET=0;0;TRUETYPE=2;DRIVER=WINSPOOL
-:close (30)
```

Format 3: *Open for [WDX]*

OPEN [INPUT] (chan[,fileopt])"[WDX]*WINPRT*[:Q_name[Q_options]]"

On an Windows Server, if you include **[WDX]** in your **OPEN [INPUT]** directive (e.g., `OPEN (30) " [WDX] *WINPRT* "`), that signals ProvideX to direct any print jobs and dialogues to the WindX client PC, which will in turn use its Windows print subsystem to send jobs to the given printer.

If you are using ***WINPRT*** on an Windows Server and do not use **[WDX]** in your **OPEN** directive, then the printer selection dialogue will appear on the server console, and any print queue you name directly must exist on the Windows Server in the Control Panel printers folder.



Reminder: You must install and use WindX to use ***WINPRT*** in a UNIX environment; however, you do not need the **[WDX]** tag on a UNIX server because ProvideX automatically directs ***WINPRT*** access to your WindX client PC.

See Also

[WINPRT_SETUP Directive, p.376](#)
[MXC \(\) / MXL \(\) Functions, p.488](#)
['FONT' Mnemonic, p.609](#)
[Printing, User's Guide](#)
[\[WDX\] Tag, p.801.](#)



*XML

XML Interface

Formats *Initializes XML Object:* **DEF OBJECT** *obj_id*,"*XML"

Where:

***XML** Keyword used in defining an XML object.

obj_id Numeric variable that will be used to save the object reference.

Description Use this interface for accessing, parsing and serializing XML documents based on the XML DOM (Document Object Model).



Note: This implementation requires activation of ProvideX XML support. Refer to the ProvideX website for licensing information.

This also requires installation of the *Xerces XML Library* as well as libraries `pvxxml.dll` (for Windows) and `pvxxml.so` (on UNIX/Linux). **TCB(193)** can be used to determine the availability of XML support on a system.

In addition, the following directives are supported for use with XML objects:

DROP OBJECT *obj_id*

DELETE OBJECT *obj_id*

The interface supports the following syntax options for accessing and manipulating XML documents:

XML*CREATE (*filename* \$, *options* \$[,*ERR=stmtref*])

Creates and opens an XML document with a given file name and document root.

Where:

filename \$ XML document file name

options \$ Control options for operation of the interface. Case-insensitive, semi-colon delimited, formatted string. Options include:

API= {DOM|SAX} - default is DOM (SAX interface not implemented).

SOURCE= {FILE|STRING} - default is FILE.

DOC_ROOT=*root_node* \$ - for creating a new XML document.

OVERWRITE= [0|1] - for overwriting an XML document, default is 0.

The document is created with ISO-8859-1 encoding. If an error occurs on executing XML functions, `PRINT MSG(-1)` can be used to view a detailed error message. Generates the following return codes:

- 1 - Opened successfully
- 0 - General error
- 1 - Warning occurs when parsing XML document
- 2 - Error occurs when parsing XML document
- 3 - Fatal error occurs when parsing XML documents

XML'OPEN (*text\$, options\$[,ERR=stmtref]*)

Opens an XML documents. When creating a new XML document, if DOC_ROOT is not specified, an error will return. If OVERWRITE is set and DOC_ROOT is set, an existing XML document will be cleared.

Where:

text\$ String variable containing either the XML document file name or the XML text. If DOC_ROOT is set, and file does not exist, a new XML document will be created with the specified filename.

options\$ Control options for operation of the interface (same as **XML'CREATE**).

Generates the same return codes as **XML'CREATE**.

XML'SET_ELEMENT (*name\$, value\$, valueType, setMode, matchValue[,ERR=stmtref]*)

Sets the current element for reading and writing.

Where:

name\$ Element tag name or attribute name.

value\$ Element value or attribute value.

valueType 1 for attribute, 2 for element tag.

setMode 1 for set to parent, 2 for set to child in the immediate sub level, or 3 for set to sibling.

matchValue 0 to ignore both *name\$* and *value\$*, 1 to match *name\$* only, 2 to match both *name\$* and *value\$*, 3 to match only the *value\$*.

Generates return codes:

- 1 - Set element successful
- 0 - Set element failed

XML'READ_ELEMENT\$ (*returnfield[,ERR=stmtref]*)

Sets the current element for reading and writing. Returns the value string.

Where:

returnfield 1 for value of current element (default), 2 for name of current element .

XML'NEXT_SIBLING ()

Sets the current element to the next sibling. Generates return codes:

- 1 - Set element successful
- 0 - Set element failed

XML'PREVIOUS_SIBLING ()

Sets the current element to the previous sibling. Generates return codes:

- 1 - Set element successful
- 0 - Set element failed

XML'READ_CHILDELEMENT\$ (*tag_name\$* [,ERR=*stmtref*])

Read the value of the child element specified by *tag_name\$* (sub-element tag). This returns the value string, or empty string if it is not found.

XML'READ_ATTRIBUTES\$ (*attribute_name\$* [,ERR=*stmtref*])

Read the value of the attribute in current element specified by *attribute_name\$*. This returns the value string, or empty string if it is not found.

XML'BUILD (*parent_tag\$, value\$* [,ERR=*stmtref*])

Initializes output buffer for a new data block with the specified parent tag as a child element of the current element.

Where:

parent_tag\$ Name of the parent tag for the new data block.

value\$ Value of the parent element.

If **XML'BUILD** is called before **XML'COMMIT** is performed, all previous data is lost.

XML'ADD_CHILDELEMENT (*tag_name\$, tag_value\$* [,ERR=*stmtref*])

Add a new child element to the end of current element block set in the output buffer.

Where:

tag_name\$ Name of the child node to be added.

tag_value\$ Value of the child node.

If output buffer has not been initialized using **XML'BUILD()**, an error will be reported.

XML'ADD_ATTRUBUTE (*attribute_name\$, attribute_value\$* [,ERR=*stmtref*])

Adds an attribute to the latest node, either created by **XML'BUILD()** or **XML'ADD_CHILDELEMENT()**.

Where:

attribute_name\$ Name of the attribute to be added.

attribute_value\$ Value of the attribute.

XML'COMMIT()

Commits the output buffer to the XML document file/string. If a buffer has not been initialized using **XML'BUILD()**, an error will be reported.

XML'CLOSE()

Closes the XML document, cleans-up workspace and releases any resources that are used. Generates return codes:

- 1 - Close failed
- 0 - Close successful

Example

Following is an example of the XML object in use for reading/writing XML documents.

```
0010 ! ---- CREATE XML DOCUMENT ----
0020 DEF OBJECT X,"*XML"
0030 X'CREATE("ProvideX.xml","DOC_ROOT=products;OVERWRITE=1")
0040 X'BUILD("product","")
0050 X'ADD_ATTRIBUTE("version","8.20")
0060 X'ADD_CHILDELEMENT("name","ProvideX")
0070 X'ADD_CHILDELEMENT("company","Sage Software")
0080 X'COMMIT()
0090 X'CLOSE()
0100 ! ---- OPEN XML DOCUMENT ----
0110 X'OPEN("ProvideX.xml","")
0120 X'SET_ELEMENT("version","8.20",1,2,1)
0130 PRINT "Company name is ",X'READ_CHILDELEMENT$("company")
0140 PRINT "Product name is ",X'READ_CHILDELEMENT$("name")
0150 PRINT "Product version is ",X'READ_ATTRIBUTE$("version")
0160 X'CLOSE()
```


Special Command Tags

Overview

Special command file tags (enclosed in square brackets) are used to modify paths and filenames for specific I/O uses in ProvideX. They may include a series of semicolon-separated parameters to be processed at run time. Some of these tags are supported only in specific operating environments.



Important: The square brackets enclosing special tags are *part of the syntax*. Other square brackets in syntax examples indicate that elements are optional.

This chapter covers the following special command tags:

[DB2] DB2 Support, p.770	Windows & WindX.
[DDE] Dynamic Data Exchange, p.776	Windows & WindX.
[DLL] Custom File Access, p.778	Windows & WindX.
[LIB] Program Library, p.781	Windows only.
[MYSQL] MySQL InnoDB Support, p.783	All platforms
[OCI] Connect to Oracle Server, p.786	Windows, some UNIX/Linux.
[ODB] Open DataBase, p.791	Windows & WindX.
[RPC] Remote Process Control, p.797	All platforms
[TCP] Transmission Control Protocol, p.799	All platforms
[WDX] Direct Action to Client Machine, p.801	Any platform via WindX.



Note: **[WDX]** specialty commands can only be used when running under WindX. You can also prefix some (but not all) of the other specialty command tags with a **[WDX]** tag to direct the action to the WindX client machine instead of the server.

WindX, the ProvideX thin-client GUI screen handler, is used to give a Windows GUI look to character-based applications, optimize a network's use of different platforms, and provide remote processing through serial and/or TCP/IP communications.



[DB2] Tag

DB2 Support

Format

OPEN (*chan*[,*fileopt1*])"**[DB2]***database*[:*table*][:*fileopt2*]"

Where:

[DB2] File tag clause to inform ProvideX that it will be opening a DB2 database (not a ProvideX data file).

chan Channel or logical file number to open.

database Name of the DB2 database to connect to.

fileopt1 File options. Supported options include:
BSZ=*num* Buffer size (in bytes)
ERR=*stmtref* Error transfer
IOL=*iolref* Default IOList
OPT=*string*\$ Open parameters (See [\[DB2\] OPT= Parameters](#))
REC=*string*\$ Record prefix (**REC=VIS**(*string*\$) can also be used).

fileopt2 **OPT=***string*\$ Open parameters (See [\[DB2\] OPT= Parameters](#))

table Name of the table to open. If the table name is not supplied, then SQL statements sent to the database must be created by the application and sent via one of the following commands:

WRITE (*chan*) **SQL**\$

WRITE RECORD (*chan*) **SQL**\$

READ (*chan*, **KEY="!**, **SQL**\$)

See [DB2/ODB Table and Column Information, p.775](#).

Description

The **[DB2]** tag is used as a prefix in an **OPEN** statement to denote that ProvideX is to route all file I/O requests to an external DB2 database server. Once you open a channel for **[DB2]** use, you can use it just like any other channel (i.e., for file I/O). It remains open until you close it. Use **TCB(198)** to check if DB2 is supported on a platform.



Note: This feature requires activation of ProvideX DB2 support. Refer to the ProvideX website for licensing information.

[DB2] OPT= Parameters

The following **OPEN** parameters can be used for connecting via DB2. This list also indicates which parameters are supported for use in the INI file ([DB2] section).

"ACCESS=" Determines type of file access required (**READ** or **WRITE**). Default is **ACCESS=WRITE**. (*INI supported*)

"AUTOCOMMIT=" Determines auto commit functionality of the database driver (either **ON** or **OFF**). (*INI supported*) It is applicable only if the driver supports transactions.

- "CONCURRENCY="** Determines the type of con-current access control/locking to be used. (*INI supported*) **READONLY** sets the cursor is set to read only - no updates allowed. **LOCK** applies low-level record locking. **OPT_VERSION** causes optimistic locking with the database version control to be used. **OPT_VALUE** causes optimistic locking with comparing record/column values to be used.
- "COMPLETE="** Determines the response to incomplete information by the following values:
- 0** (`SQL_DRIVER_NOPROMPT`). *Default.* Driver Manager copies the connection string specified by the application.
 - 1** (`SQL_DRIVER_COMPLETE`) or
 - 3** (`SQL_DRIVER_COMPLETE_REQUIRED`). If the connection string specified by the application includes the DSN keyword, the Driver Manager copies the connection string specified by the application. Otherwise, it takes the same actions as `SQL_DRIVER_PROMPT`.
 - 2** (`SQL_DRIVER_PROMPT`). If connection string does not contain either `DRIVER`, `DSN`, or `FILEDSN` keyword, the Driver Manager displays the Data Sources dialog box. It constructs a connection string from the data source name returned by the dialog box and any other keywords passed to it by the application. If the data source name returned by the dialog box is empty, the Driver Manager specifies the keyword-value pair **DSN=Default**. (This dialog box will not display a data source with the name "Default".)
- All options except `NOPROMPT` require the handle of the parent window, which will be the handle of the currently active ProvideX window. This parameter is not used if a connection string is not supplied.
- "CONNECT="** Specifies a connection string surrounded by a delimiter character, enabling use of a "dsn-less" connection. Connection strings are driver specific. Consult the driver's reference for supported connection string values. Under UNIX/Linux, this parameter requires **COMPLETE=0** (see above).
- If a connection string is supplied the value of the database name is ignored. If the database name is not null then the value is used only for determining if a connection should be shared. For example,
- ```
Open(1,iol=*,opt="connect='DSN=nomads' ") "[DB2]foo;Customer"
Open(2,iol=*,opt=" ") "[DB2]foo;Customer Classes"
Open(3,iol=*,opt=" ",err=*next) "[DB2];Customer"
Open(4,iol=*,opt="connect='DSN=foo' ") "[DB2]foo;Customer"
```
- The table opened on channel 2 will share the connection because of `foo`. Channel 3 will error-out because neither a valid database name nor a valid connect string was supplied. Channel 4 will open the table `Customer` using the properties of the "nomads" DSN because `foo` matches (the connect string was ignored).



If the keywords **USER=** or **PSWD=** are supplied on the open then the values of these attributes will be appended to the connection string.

- "CURSOR\_TYPE="** Defines the type of cursor that is to be used. **FORWARD** indicates that any result sets can be read in a forward only direction. (*INI supported*) **STATIC** indicates that the result set is static. **KEYSET** forces the cursor to use/maintain record keys in a Keyset. **DYNAMIC** indicates that the cursor is effective in the current Rowset only.
- "CURSOR\_USE="** Defines the type of cursor to be used. (*INI supported*) **DRIVER** (default) assumes the specific driver's own cursors. **ODBC** causes the ODBC interface to use the "Driver Managers" cursor library that may provide additional functionality not available within the database driver. **IF\_NEEDED** tells the system to use the specific database driver's own cursor functionality unless the additional functionality is requested specifically.
- "DATEFMT="** Date format mask applying to all date fields in table. (*INI supported*) This can be a combination of Y M D with any other characters; e.g., to convert dates to 4-character year, month and day: DATEFMT=YYYYMMDD. Other characters are inserted as is; e.g., DATEFMT=YY/MM/DD with a date of March 1, 2004 would be returned as 04/03/01.
- Two packed century formats are also supported. The first format, AA, maps A to 2000. Our example of March 1, 2004 with DATEFMT=AAMMDD would be returned as A40301. The second format, KK, is similar except K maps to 2000. A DATEFMT=KKMMDD would return K40301 for the example.
- "DB="** or **"QUALIFIER="** Qualifies the specific database that you wish to use when using a driver to service multiple databases. (*INI supported*)
- "DEBUGIT="** String to append to SQL statement along with program name and line number for debugging purposes. (*INI supported*) This must indicate the comment character(s) appropriate to the database. For example, "--" is the comment identifier for Microsoft SQL Server; anything after "--" is ignored by SQL Server when compiling the SQL statement.
- "EXEC\_SPRNO="** Name of stored procedure used to emulate **RNO( )** function.
- "EXTROPT="** Controls the format of the **SELECT** statement used to process an **EXTRACT**. (*INI supported*) By default, PVX generates a **SELECT \* FROM table FOR UPDATE WHERE . . .**
- When **EXTROPT=text**, then *text* is substituted in place of **FOR UPDATE**. In addition, if the first character of *text* is \$, then the remaining characters of *text* are placed at the end of the **SELECT** statement rather than after the filename. This allows for different variations of **SQL** to be supported.
- "IND="** Identifies a column that contains a sequential number starting at 0. This is used to emulate an indexed file.

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "ISOLATION="   | Controls the isolation that this connection will have relative to other processes on the same database. In particular, it controls <i>Dirty</i> reads (reading data that may be rolled back), <i>Non-Repeatable</i> reads (reading data after being changed by other transactions), and <i>Phantom</i> reads (reading data newly added to file). ( <i>INI supported</i> )<br><br>Settings include: UNCOMMITTED ( <i>D, R, P</i> possible), COMMITED ( <i>D</i> possible, <i>R &amp; P</i> not possible), REPEATABLE ( <i>P</i> possible, <i>D &amp; R</i> not possible), SERIAL ( <i>D, R, &amp; P</i> not possible). |
| "KEY="         | Identifies fields that make up the key(s). For named keys enter <b>*NAME:keyname</b> ; e.g.,<br><br><b>OPEN(chan)"[ODB]dsn;table;KEY=field,field,*NAME:keyname"</b><br><br>Use the <b>:D</b> option to indicate that the key segment is to be sorted in descending order; e.g., <b>KEY=KeyFld1 , KeyFld2 : D , KeyFld3.</b>                                                                                                                                                                                                                                                                                           |
| "KEYDATA="     | Identifies a column that represents the key. This is used to emulate an external key where the data is not duplicated in the data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| "KEYSET_SIZE=" | Size of the Keyset for use with the cursor. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| "MAS90DATE"    | Reformats the contents of a date column to and from the Sage MAS 90 date format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| "MAS90SET"     | Sets flags for Sage MAS 90 emulation, such as turning on the <b>MAS90DATE</b> conversion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| "MAXROWS="     | Maximum number of rows/records returned. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| "NONUMADJ="    | Set to <b>1, Y</b> or <b>y</b> to suppress +3 adjustment for defined length of numerics. ( <i>INI supported</i> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| "NONULLS="     | Inserts zero-length strings rather than nulls into the target database, and does not generate <b>WHERE</b> clauses checking for <b>IS NULL</b> or <b>IS NOT NULL</b> . ( <i>INI supported</i> ) Set to <b>1, Y</b> or <b>y</b> to enable or <b>0, N</b> or <b>n</b> to disable. If the application does not work correctly when moving from Version 5 or lower, then set <b>NONULLS=P</b> to indicate that keys are handled the same as pre-Version 6.                                                                                                                                                                |
| "NOSTRIP"      | Keeps trailing spaces (Default)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| "NULLPADKEY"   | Forces keys to be padded to full length with the null character, <b>\$00\$</b> . When used in an INI file, set <b>NULLPADKEY=1, Y</b> or <b>y</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| "POSUPDATE="   | Determines use of <i>SqlSetPos</i> functions. ( <i>INI supported</i> ) Use one of the following: <b>M</b> (must use positioned update), <b>O</b> (default, optionally use positioned update), <b>N</b> (never use positioned update).                                                                                                                                                                                                                                                                                                                                                                                 |
| "PREPARE="     | Set to <b>1, Y</b> or <b>y</b> to use prepared statements. ( <i>INI supported</i> ) Prepared statements are pre-compiled SQL that may improve performance.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| "PSWD="        | Specifies password. ( <i>INI supported</i> , but <i>not secure</i> ). Anyone with access to the INI will be able to read this password.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "REC="         | Provides the column names, type, and size. This is typically done to improve performance. If this information is not provided, then ProvideX must query the database for this information. For more information, see <a href="#">ODB/OCI/DB2 Record Processing, p.789</a> .                                                                                                                                                                                                                                                                         |
| "RECDATA="     | Identifies a column to return as the full record. This can be used for variant records which use complex rules to identify the record type.                                                                                                                                                                                                                                                                                                                                                                                                         |
| "ROWSET_SIZE=" | Size of the Rowset used by the cursor. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| "SHARED"       | Sets all tables to share a single connection to the Oracle database. Default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| "STDDATE"      | Overrides the above formatting on individual columns.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| "STRIP"        | Removes trailing spaces from fields                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| "TEXTMAX="     | Overrides maximum size for text fields (default is 4096 bytes). ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| "TIMEOUT="     | Defines the time out value for any SQL operation (time before error 0 returned). ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| "TOP="         | Specifies use of the TOP clause in SELECT statements (limits the number of rows to return in a result set). ( <i>INI supported</i> ) If <b>TOP=n</b> is non-zero, then the <b>KEF()</b> / <b>KEL()</b> functions issue a <b>SELECT TOP 1 . . . SQL statement</b> , which improves system performance. If <b>TOP=n &gt; 0</b> , then PVX issues <b>SELECT TOP n</b> to reduce the data transferred. <b>TOP=-1</b> indicates the driver supports <b>SELECT TOP</b> , but normal reading should not use it. Default is 0 ( <i>TOP not supported</i> ). |
| "TSQL="        | Defines a SQL statement that is used to control what data the logical file returns.                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| "TYP="         | Sets identifier for different variant records. For more information see <a href="#">ODB/OCI/DB2 Record Processing, p.789</a> .                                                                                                                                                                                                                                                                                                                                                                                                                      |
| "UNIQUE"       | Sets new opens to be on a unique connection to the database. ( <i>INI supported</i> ) When used in an INI file, set <b>UNIQUE=1, Y or y</b> . <b>UNIQUE=0, N</b> or <b>n</b> indicates a shared connection.                                                                                                                                                                                                                                                                                                                                         |
| "USER="        | Specifies login name. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

**Example**

To open a connection to a DB2 table using the column names to generate the IOList:

```
OPEN (14, IOL=*) "[DB2]DB2instance,DB2table"
```

**See Also**

[READ Directive, p.271](#),  
[READ RECORD Directive, p.275](#),  
[SELECT.FROM..NEXT RECORD Directive, p.299](#),  
[WRITE Directive, p.383](#),  
[WRITE RECORD Directive, p.386](#)  
[OPEN Directive, p.232](#)

## DB2/ODB Table and Column Information

When accessing an external database with a raw (no table specified) connection, it is possible to find out what the table, columns, and indices are. This is achieved using a **READ** and the following **KEY=** values.

**KEY="?"**

Results in an `SQLTables()` call with null parameters.

**KEY="\*Table\$"**

The asterisk will be stripped and whatever value is in `Table$` will be passed as the table parameter of a `SQLColumns()` call.

**KEY="\*\*Table\$"**

The asterisks will be stripped and whatever value is in `Table$` will be passed as the table parameter of a `SQLStatistics()` call with the attribute `SQL_INDEX_ALL`.

**Example:**

```
OPEN (chan) "[DB2]Database; ;"
READ (chan,KEY="?") IOL=TableIOList
```



## [DDE] Tag

## Dynamic Data Exchange

Format `OPEN (chan[,fileopt])[DDE]dde_app;params"`

Where:

**[DDE]** File tag clause to inform ProvideX that it will be opening an external *Dynamic Data Exchange* (DDE) application.

*chan* Channel or logical file number to open.

*dde\_app* Path and/or name of DDE application; e.g., `Excel`. String expression.

*fileopt* File options.

*params* Optional DDE-specific parameters. Semicolon-separated arguments and/or variables to receive returned values, etc. (Early implementations of ProvideX used a vertical bar instead of a semicolon as separator — both are now acceptable).

Please refer to the documentation supplied with the individual product or application to determine how to communicate with it using the **[DDE]** link.



*Note:* For use in *WindX* or *Windows* only.

### Description

The **[DDE]** tag is used as a prefix in an **OPEN** statement to denote that ProvideX is to route all file I/O requests to an external DDE application; e.g., `Excel`. ProvideX recognizes the tag and deals with it internally at run time.

You can associate a CTL value with an item in a DDE application to have ProvideX generate the CTL value automatically whenever the value of the associated item changes; e.g.,

```
0010 DEFCTL (dde_fileno)"item"=ctl_event
```

### Troubleshooting

In the event that the DDE server is unavailable, ProvideX will attempt to start the application and establish a DDE connection to it. The following paths are used to locate the application:

- Directory from which the application loaded
- Current directory
- Windows system directory
- Windows directory
- All directories listed in the `PATH` environment variable.

If these search rules cannot locate the application, then an `Error #10: Illegal pathname specified` will be reported, and **MSG(-1)** will contain text similar to `"DdeConnect Failed (err/ret=2/16394)."`



However, this situation can be prevented:

- By adding the path to the executable to the PATH environment variable
- By pre-launching the application prior to opening the DDE connection; e.g.,  
SYSTEM\_HELP "Excel.exe".



**Note:** SYSTEM\_HELP has an advantage over the INVOKE directive because it uses different Windows API calls that are better for locating the program in question.

## Example

This example illustrates an export to an Excel spreadsheet. Note that the worksheet name is optional, but the worksheet must exist if you include its name. This example also demonstrates the use of \$09\$, **Tab**, as the separator for Excel and makes selective graphical requests to draw a pie chart, etc.:

```
0010 OPEN (1)"[WDX][DDE]excel;existing_worksheet.wk1"
! or 0010 OPEN (1)"[wdx][dde]Excel;"
0020 OPEN (2)"SALES"
0030 LET R=0
0040 LOOP:
0050 LET DIV_ID$=KEY(2,END=DRAW_IT)
0060 READ (2,KEY=DIV_ID$)DIV_NAME$,DIV_SALES
0070 LET R=R+1 ! Bump row number
0080 LET K$="R"+STR(R)+"C1:R"+STR(R)+"C2"
0090 WRITE RECORD (1,KEY=K$)DIV_NAME$+09+STR(DIV_SALES)
0100 GOTO LOOP
0110 DRAW_IT:
0120 IF R=0 THEN STOP ! No divisions
0130 LET X$="R"+STR(R)
0140 WRITE RECORD (1)"[select("R1C1:"+X$+"C2",""+X$+"C2")]"
0150 WRITE RECORD (1)"[new(2,1)]"
0160 WRITE RECORD (1)"[gallery.3d.pie(6)]"
0170 WRITE RECORD (1)"[window.maximize()]"
0180 WRITE RECORD (1)"[app.maximize()]"
```

## See Also

[WRITE RECORD Directive, p.386](#)

[OPEN Directive, p.232](#)



## [DLL] Tag

## Custom File Access

Format `OPEN (chan,[fileopt])"[DLL:lib_name;fnc_name]params"`

Where:

|                 |                                                                                                             |
|-----------------|-------------------------------------------------------------------------------------------------------------|
| <b>[DLL ..]</b> | File tag clause to inform ProvideX that it will be opening an external DLL for I/O requests.                |
| <i>chan</i>     | Channel or logical file number to open.                                                                     |
| <i>fileopt</i>  | File options.                                                                                               |
| <i>fnc_name</i> | Case-sensitive name of the function. It acts as the entry point into the library. String expression.        |
| <i>lib_name</i> | Path and/or name of the DLL file that contains the external function you want to invoke. String expression. |
| <i>params</i>   | DLL-specific parameters. Semicolon-separated arguments and/or variables to receive returned values, etc.    |



*Note:* For use in *WindX* or *Windows* only.

### Description

The **[DLL]** tag is used as a prefix in an **OPEN** statement to denote that ProvideX is to route all file I/O requests via an external DLL (*Dynamic Link Library*) file. This is intended primarily for mapping to customized (user-defined) routines for file access. For access methods in ProvideX using industry-standard mechanisms, refer to the sections on the **[OCI]** and **[ODB]** tags.

### Calling Sequence

This interface uses a single parameter block to handle all communication between ProvideX and the DLL. Whenever a DLL entry point is called, the parameter block is passed to it as its argument.

The DLL's return value determines completion status. A successful completion returns -1. Any other termination status is assumed to be a ProvideX error code; e.g.,

|    |                  |
|----|------------------|
| 0  | Record/File Busy |
| 2  | End of file      |
| 11 | Record not found |
| 12 | File not found   |

The parameter block is defined as follows:

```
struct IODLL_PARAM
{
 long handle; /* DLL specific handle */
 int nFunction; /* IO Function */
 int nMode; /* Access mode (Next, KEY=, IND=, ...) */
 int lenBuffer; /* Size of buffer */
 char *pBuffer; /* Pointer to buffer */
 int lenRecID; /* Length or key of index/rec# */
 char *pRecID; /* Pointer to KEY buffer*/
 int idxRecID; /* Key Number/Index Number/RNO (base 0) */
 int nKNOValue; /* KNO= Specified on directive/function */
};
```

**Where:**

- handle** Identifies the specific file for the request. On the **OPEN**, the DLL returns this value, which is used for all subsequent calls.
- nFunction** Identifies the type of function being performed (**OPEN**, **CLOSE**, ...). Possible values include:
- ```
#define IODLL_FNC_OPEN      1
#define IODLL_FNC_CLOSE    2
#define IODLL_FNC_READ     3
#define IODLL_FNC_EXTRACT  4
#define IODLL_FNC_WRITE    5
#define IODLL_FNC_INSERT   6
#define IODLL_FNC_REMOVE   7
#define IODLL_FNC_KEYNEXT  8
#define IODLL_FNC_KEYPREV  9
#define IODLL_FNC_KEYCUR   10
#define IODLL_FNC_KEYFIRST 11
#define IODLL_FNC_KEYLAST  12
#define IODLL_FNC_RNO      13
#define IODLL_FNC_IND      14
#define IODLL_FNC_LOCK     15
#define IODLL_FNC_UNLOCK   16
#define IODLL_FNC_PURGE    17
#define IODLL_FNC_KEN      18

#define IODLL_FNC_LOAD     -1
#define IODLL_FNC_FREE     -2
```
- nMode** Identifies the access method being applied during a **READ/WRITE** operation. Possible access values include:
- ```
#define IODLL_MODE_NEXT 0
#define IODLL_MODE_BY_KEY 1
#define IODLL_MODE_BY_IND 2
#define IODLL_MODE_BY_RNO 3
```
- lenBuffer** Length of the data buffer used for **READ/WRITE**.
- pBuffer** Pointer to the data buffer to be used for **READ/WRITE** operations.

- lenRecID* Length of the key field for **READ/WRITE/REMOVE** operations when accessing via key or requesting a key.
- pRecID* Pointer to the data buffer to hold the key.
- idxRecID* The record index number when accessing record index, record number, or the key number.
- nKNOValue* **KNO=** specified on a directive/function (-1 if not specified).

### Example

```
OPEN (1)"[dll:server.dll;EntryPoint]filename"
OPEN (1)"[dll:dbase2.dll;entry]cust.db"
```

### See Also

[OPEN Directive, p.232](#)  
[\[OCI\] Tag, p.786](#)  
[\[ODB\] Tag, p.791](#)  
[DLL\(\) Function, p.418](#)



## [LIB] Tag

## Program Library

### Format

1. Define Search Rules: **PREFIX PROGRAM** "[LIB:proglib]"
2. Write to File: **SAVE** "[LIB:proglib]prog"
3. Read into Memory: **LOAD** "[LIB:proglib]prog"
4. Change Name: **RENAME** "[LIB:proglib]prog" **TO** ...
5. Delete from System: **ERASE** "[LIB:proglib]prog"

### Where:

- [LIB: ..]** File tag clause informs ProvideX that a file belongs to the program library, *proglib*.
- prog* Name of program.
- proglib* Path and filename of program library.



**Note:** For use in *Windows* only. Program libraries cannot be accessed remotely; i.e., the [WDX] tag is not supported.

### Description

The [LIB] tag is used as a prefix to denote that ProvideX is to access programs in a *program library*, a single keyed file/library where each record contains the object for a program stored within.

Saving/loading programs from a single file reduces OS file searching and security checking, and can improve system performance. Program libraries also make it easier for application developers to ship and install applications. Fundamentally, program libraries are transparent to applications and are handled much the same way as directories.

The pathname for a program library is indicated as part of the prefix tag, following the colon: [LIB:*proglib*]. The actual program name follows the prefix. For example, if the program PROG01 is in the library /usr/myappl/proglib it would be referenced as [LIB:/usr/myappl/proglib]PROG01.

To simplify access to libraries, they can also be defined in a **PREFIX** (generally a program prefix); e.g., **PREFIX PROGRAM** "[LIB:/usr/myappl/proglib]".

### Cached Libraries

The system automatically maintains a list of open program libraries. Libraries are kept open if any programs within a library are in use. In addition, the system maintains a cache of opened program libraries. By default the number of cached program libraries is 10; however, this is alterable by setting the '**PL**'= [System Parameter, p.679](#). Program libraries would be closed on a **START, QUIT**, or whenever '**PL**' is changed.

### *Adding, Changing, or Removing Programs*

Access to programs within a program library is handled via the standard **SAVE**, **LOAD**, **RENAME**, and **ERASE** directives. A **SAVE** creates a new record in the library. When addressing an existing program, **SAVE** replaces the record containing the program image in the library with new program contents. The **LOAD** command reads the records from the library. The **ERASE** command delete records from the library.

The **RENAME** directive can rename a program in a library. It does not allow a program in a library to be renamed into another library or to a stand-alone program, or vice-versa. In **RENAME** syntax, the original (*name1*) can be defined as a "file within a program library", the new *name2* is assumed to be its new name in the library.

### *Creating Program Library Files*

The program library file is automatically created the first time you save a program to it. For example, `SAVE "[lib:proglib]myprog"` creates a program library called `proglib` as a keyed file with the following characteristics:

```
Maximum record size : 30000 (Variable)
Maximum # of records : (No limit)
Size of key block : 30720 bytes
Record Expansion factor : 10%
Extended attributes : Extended records
External key size : 0
Alt. key 0 : [0:1:32:"C"]
```

Alternate keys can be defined if desired for program info; e.g.,

```
0:51:12 Saved user name
0:103:4 Save time in binary
0:115:2 Owner id in binary
```

The file can be encrypted; however, pre-open the application and provide its password before trying to use it so that its password can be cached.

The actual keys used in a program library will be subject to the same rules as normal path names with regards to the **'FU'** and **'FL'** system parameters. If **'FU'** is set then the file names (key value) will be converted to upper case. If **'FL'** is set the file names (key value) will be converted to lower case.

In addition, the directory delimiters `"/"` and `"\"` will both be converted to `"/"` thus allowing applications to be portable between operating systems.

# [MYSQL] Tag

# MySQL InnoDB Support

## Format

**OPEN** (*chan[,fileopt]*)"[MYSQL]host;database[;table][;fileopt]"

Where:

**[MYSQL]** File tag clause to inform ProvideX that it will be opening a MySQL database (not a ProvideX data file).

*chan* Channel or logical file number to open.

*host* MySQL database host name (IP address or DSN name).

*database* Name of the MySQL database to connect to.

*fileopt1* File options. Supported options include:

**BSZ=num** Buffer size (in bytes)

**ERR=stmtref** Error transfer

**IOL=iolref** Default IOList

**OPT=string\$** Open parameters (See [\[MYSQL\] OPT= Parameters](#))

**REC=string\$** Record prefix (**REC=VIS(string\$)** can also be used).

*table* Name of the table to open. If the table name is not supplied, then SQL statements sent to the database must be created by the application and sent via one of the following commands:

**WRITE** (*chan*) SQL\$

**WRITE RECORD** (*chan*) SQL\$

**READ** (*chan*, KEY="!", SQL\$)

See [DB2/ODB Table and Column Information, p.775](#).

## Description

The **[MYSQL]** tag is used as a prefix in an **OPEN** statement to denote that ProvideX is to route all file I/O requests to an external MySQL database server. Once you open a channel for use, you can use it just like any other channel (i.e., for file I/O). It remains open until you close it. Use **TCB(194)** to check if MySQL is supported on a platform.



**Note:** This feature requires activation of ProvideX MySQL support. Refer to the ProvideX website for licensing information.

## [MYSQL] OPT= Parameters

The following **OPEN** parameters can be used for connecting via MySQL. This list also indicates which parameters are supported for use in the INI file ([MYSQL] section).

**"AUTOCOMMIT="** Determines auto commit functionality of the MySQL database. Only the *InnoDB* storage engine is supported. (*INI supported*)

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "DATEFMT="    | Date format mask applying to all date fields in table. ( <i>INI supported</i> ) This can be a combination of Y M D with any other characters; e.g., to convert dates to 4-character year, month and day: DATEFMT=YYYYMMDD. Other characters are inserted as is; e.g., DATEFMT=YY/MM/DD with a date of March 1, 2004 would be returned as 04/03/01.<br><br>Two packed century formats are also supported. The first format, AA, maps A to 2000. Our example of March 1, 2004 with DATEFMT=AAMMDD would be returned as A40301. The second format, KK, is similar except K maps to 2000. A DATEFMT=KKMMDD would return K40301 for the example. |
| "DEBUGIT="    | String to append to SQL statement along with program name and line number for debugging purposes. Refer to MySQL documentation for comment styles. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| "EXTROPT="    | Controls the format of the SELECT statement used to process an EXTRACT. ( <i>INI supported</i> ) By default, PVX generates a SELECT * FROM <i>table</i> FOR UPDATE WHERE . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| "FACTSDT"     | Causes all date fields to be translated to/from the SQL date format to the format used by the FACTS application                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| "IND="        | Identifies a column that contains a sequential number starting at 0. This is used to emulate an indexed file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| "KEY="        | Identifies fields that make up the key(s).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| "KEYDATA="    | Identifies a column that represents the key. This is used to emulate an external key where the data is not duplicated in the data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| "LOCKTABLES=" | Determines if a MySQL LOCK TABLES statement or ProvideX prefix file locking is used to lock tables. The default is using prefix file locking. If a LOCK TABLES statement is used, a table locked by a third party will cause ProvideX to wait for the table to be released or until the MySQL connection is timed out.                                                                                                                                                                                                                                                                                                                      |
| "MAS90DATE"   | Reformats the contents of a date column to and from the Sage MAS 90 date format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| "MAS90SET"    | Sets flags for Sage MAS 90 emulation, such as turning on the MAS90DATE conversion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| "NONUMADJ="   | Set to 1, Y or y to suppress +3 adjustment for defined length of numerics. ( <i>INI supported</i> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| "NONULLS="    | Inserts zero-length strings rather than nulls into the target database, and does not generate WHERE clauses checking for IS NULL or IS NOT NULL. ( <i>INI supported</i> ) Set to 1, Y or y to enable or 0, N or n to disable. If the application does not work correctly when moving from Version 5 or lower, then set NONULLS=P to indicate that keys are handled the same as pre-Version 6.                                                                                                                                                                                                                                               |
| "NOSTRIP"     | Keeps trailing spaces (Default)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| "NULLPADKEY"  | Forces keys to be padded to full length with the null character, \$00\$.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |



|            |                                                                                                                                                                                                                                                                             |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "PSWD="    | Specifies password. ( <i>INI supported</i> , but <i>not secure</i> . Anyone with access to the INI will be able to read this password.)                                                                                                                                     |
| "REC="     | Provides the column names, type, and size. This is typically done to improve performance. If this information is not provided, then ProvideX must query the database for this information. For more information, see <a href="#">ODB/OCI/DB2 Record Processing, p.789</a> . |
| "RECDATA=" | Identifies a column to return as the full record. This can be used for variant records which use complex rules to identify the record type.                                                                                                                                 |
| "STDDATE"  | Use standard date formatting.                                                                                                                                                                                                                                               |
| "STRIP"    | Removes trailing spaces from fields                                                                                                                                                                                                                                         |
| "TEXTMAX=" | Overrides maximum size for text fields (default is 4096 bytes). ( <i>INI supported</i> )                                                                                                                                                                                    |
| "TIMEOUT=" | Defines the time out value for any SQL operation (time before error 0 returned). ( <i>INI supported</i> )                                                                                                                                                                   |
| "TOP="     | Specifies use of the LIMIT clause in SELECT statements (limits the number of rows to return in a result set). ( <i>INI supported</i> )                                                                                                                                      |
| "TSQL="    | Defines a SQL statement that is used to control what data the logical file returns.                                                                                                                                                                                         |
| "TYP="     | Sets identifier for different variant records. For more information see <a href="#">ODB/OCI/DB2 Record Processing, p.789</a> .                                                                                                                                              |
| "USER="    | Specifies login name. ( <i>INI supported</i> )                                                                                                                                                                                                                              |

**Example**

```
0010 OPEN (1,IOL=*)" [mysql]localhost;db;table;user=root;pswd=px;"
0020 READ (1)
0030 PRINT "Current key: ",KEC(1)
0040 PRINT "SQL Statement executed: ",KEN(1)
0050 CLOSE(1)
```

**See Also**

[READ Directive, p.271](#),  
[READ RECORD Directive, p.275](#),  
[SELECT..FROM..NEXT RECORD Directive, p.299](#),  
[WRITE Directive, p.383](#),  
[WRITE RECORD Directive, p.386](#)  
[OPEN Directive, p.232](#)

**[OCI]** Tag

## Connect to Oracle Server

|                 |                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Format          | <code>OPEN (chan[fileopt1])[OCI]sid[;table][;fileopt2]"</code>                                                                                                                                                                                                                                                                                                              |
|                 | <i>Where:</i>                                                                                                                                                                                                                                                                                                                                                               |
| <b>[OCI]</b>    | File tag clause to inform ProvideX that it will be opening an Oracle database.                                                                                                                                                                                                                                                                                              |
| <i>chan</i>     | Channel or logical file number to open.                                                                                                                                                                                                                                                                                                                                     |
| <i>fileopt1</i> | File options. Supported options include:<br><b>BSZ=num</b> Buffer size (in bytes)<br><b>ERR=stmtref</b> Error transfer<br><b>IOL=iolref</b> Default IOList<br><b>NBF=num</b> Dedicated number of buffers<br><b>OPT=String\$</b> Open parameters (See <b>[OCI] OPT= Parameters</b> below)<br><b>REC=string\$</b> Record prefix ( <b>REC=VIS(string\$)</b> can also be used). |
| <i>fileopt2</i> | <b>OPT=string\$</b> Open parameters (See <b>[OCI] OPT= Parameters</b> )                                                                                                                                                                                                                                                                                                     |
| <i>sid</i>      | Oracle System ID of file to open. If not supplied, then the value of the environment variable ORACLE_SID is used. String expression.                                                                                                                                                                                                                                        |
| <i>table</i>    | Name of the table to open. If the table name is not supplied, then SQL statements sent to the database must be created by the application and sent via one of the following commands:<br><b>WRITE (chan) SQL\$</b><br><b>WRITE RECORD (chan) SQL\$</b><br><b>READ (chan, KEY="!", SQL\$)</b>                                                                                |

**Description** The **[OCI]** tag is used as a prefix in an **OPEN** statement to denote that ProvideX is to route all file I/O requests to an external (not ProvideX) Oracle database. (**OCI** is an acronym for *Oracle Call Interface*.) Once you open a channel for **[OCI]** use, you can use it just like any other channel (i.e., for file I/O). It remains open until you close it.

**Note:** This feature requires activation of ProvideX OCI support (available for Windows, Redhat, HP UX, Sun Solaris, and AIX). Refer to the ProvideX website for licensing information. Use **TCB(200)** to check if OCI is supported on the platform.

**[OCI] OPT= Parameters**

The **OPEN** options for connecting to an Oracle server are listed below: "

- "AUTO\_INDEX="** Used to include hints and index numbers to **SELECT** statements
- "DATEFMT="** Date format mask applying to all date fields in table. (*INI supported*) This can be a combination of **YMD** with any other characters; e.g., to convert dates to 4-character year, month and day: **DATEFMT=YYYYMMDD**. Other characters are inserted as is; e.g., **DATEFMT=YY/MM/DD** with a date of March 1, 2004 would be returned as 04/03/01.

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                      | Two packed century formats are also supported. The first format, <b>AA</b> , maps A to 2000. Our example of March 1, 2004 with <b>DATEFMT=AAMMDD</b> would be returned as <b>A40301</b> . The second format, <b>KK</b> , is similar except <b>K</b> maps to 2000. A <b>DATEFMT=KKMMDD</b> would return <b>K40301</b> for the example.                                                                                                                                                                                                                            |
| <b>"DEBUGIT="</b>    | String to append to SQL statement along with program name and line number for debugging purposes. ( <i>INI supported</i> ) This must indicate the comment character(s) appropriate to the database. For example, "--" is the comment identifier for Microsoft SQL Server; anything after "--" is ignored by SQL Server when compiling the SQL statement.                                                                                                                                                                                                         |
| <b>"EXEC_SPRNO="</b> | Not applicable to Oracle at this time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>"EXTROPT="</b>    | Controls the format of the <b>SELECT</b> statement used to process an <b>EXTRACT</b> . ( <i>INI supported</i> ) By default, <b>PVX</b> generates a <b>SELECT * FROM table FOR UPDATE WHERE . . .</b> When <b>EXTROPT=text</b> , then <b>text</b> is substituted in place of <b>FOR UPDATE</b> . In addition, if the first character of <b>text</b> is <b>\$</b> , then the remaining characters of <b>text</b> are placed at the end of the <b>SELECT</b> statement rather than after the filename. This allows for different variations of SQL to be supported. |
| <b>"IND="</b>        | Identifies a column that contains a sequential number starting at 0. This is used to emulate an indexed file.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>"IOPROG="</b>     | Emulates the embedded I/O program logic available with a true ProvideX file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>"KEY="</b>        | Identifies fields that make up the key(s). For named keys enter <b>*NAME:keyname</b> ; e.g.,<br><b>OPEN(chan)"[OCI]sid;table;KEY=field,field,*NAME:keyname"</b><br>Use the <b>:D</b> option to indicate that the key segment is to be sorted in descending order; e.g., <b>KEY=KeyFld1,KeyFld2:D,KeyFld3</b> .                                                                                                                                                                                                                                                   |
| <b>"KEYDATA="</b>    | Identifies a column that represents the key. This is used to emulate an external key where the data is not duplicated in data.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>"MAS90DATE"</b>   | Reformats the contents of a date column to and from the Sage MAS 90 date format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>"MAS90SET"</b>    | Sets flags for Sage MAS 90 emulation, such as turning on the <b>MAS90DATE</b> conversion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>"NONUMADJ="</b>   | Set to <b>1</b> , <b>Y</b> or <b>y</b> to suppress +3 adjustment for defined length of numerics. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>"NONULLS="</b>    | Inserts zero-length strings rather than nulls into the target database, and does not generate <b>WHERE</b> clauses checking for <b>IS NULL</b> or <b>IS NOT NULL</b> . ( <i>INI supported</i> ) Set to <b>1</b> , <b>Y</b> or <b>y</b> to enable or <b>0</b> , <b>N</b> or <b>n</b> to disable. If the application does not work correctly when moving from Version 5 or lower, then set <b>NONULLS=P</b> to indicate that keys are handled the same as pre-Version 6.                                                                                           |
| <b>"NOSTRIP"</b>     | Keeps trailing spaces (Default).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "NULLPADKEY" | Forces keys to be padded to full length with null character, <b>\$00\$</b> . ( <i>INI supported</i> ) When used in an INI file, set <b>NULLPADKEY=1, Y or y</b> .                                                                                                                                                                                                                                                                                                                                                                                                        |
| "ORACLE="    | Indicates if the database uses ORACLE SQL sequence (either <b>Y or N</b> ). ( <i>INI supported</i> ) If <b>ORACLE=</b> and <b>TOP=</b> are used, then <b>SELECT</b> commands are generated as <b>SELECT * FROM (SELECT * FROM TABLE) WHERE ROWNUM &lt; 1</b> . Default is <b>Y</b> .                                                                                                                                                                                                                                                                                     |
| "PREPARE="   | Set to <b>1, Y or y</b> to use prepared statements. ( <i>INI supported</i> ) Prepared statements are pre-compiled SQL that may improve performance. Default is <b>N</b> .                                                                                                                                                                                                                                                                                                                                                                                                |
| "PSWD="      | Specifies password. ( <i>INI supported, but not secure</i> . Anyone with access to the INI will be able to read this password.)                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| "REC="       | Provides the column names, type, and size. This is typically done to improve performance. If this information is not provided, then ProvideX must query the database for this information. For more information see <b>ODB/OCI/DB2 Record Processing, p.789</b> .                                                                                                                                                                                                                                                                                                        |
| "RECDATA="   | Identifies a column to return as full record. This can be used for variant records that use complex rules to identify the record type.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| "SHARED"     | Sets all tables to share a single connection to the Oracle database (Default).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| "STDDATE"    | Overrides the above formatting on individual columns.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| "STRIP"      | Removes trailing spaces from fields                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| "TEXTMAX="   | Overrides maximum size for text fields (default is 4096 bytes). ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| "TOP="       | Specifies use of the <b>TOP</b> clause in <b>SELECT</b> statements (limits the number of rows to return in a result set). ( <i>INI supported</i> ) If <b>TOP=n</b> is non-zero, then the <b>KEF()</b> / <b>KEL()</b> functions issue a <b>SELECT</b> where the row number is $\leq n$ , which improves system performance. If <b>TOP=n &gt; 0</b> , then PVX issues <b>SELECT TOP n</b> to reduce the data transferred. <b>TOP=-1</b> indicates the driver supports <b>SELECT TOP</b> , but normal reading should not use it. Default is 0 ( <b>TOP not supported</b> ). |
| "TSQL="      | Not applicable to Oracle at this time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| "TYP="       | Sets identifier for different variant records. For more information see <b>ODB/OCI/DB2 Record Processing, p.789</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| "UNIQUE"     | Sets new opens to be on a unique connection to the database. ( <i>INI supported</i> ) When used in an INI file, set <b>UNIQUE=1, Y or y</b> . <b>UNIQUE=0, N or n</b> indicates a shared connection.                                                                                                                                                                                                                                                                                                                                                                     |
| "USER="      | Specifies login name. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |



## See Also

[READ Directive, p.271](#),  
[READ RECORD Directive, p.275](#),  
[SELECT..FROM..NEXT RECORD Directive, p.299](#),  
[WRITE Directive, p.383](#),  
[WRITE RECORD Directive, p.386](#)  
[OPEN Directive, p.232](#)

## ODB/OCI/DB2 Record Processing

The **REC=** phrase is used to control the formatting of the data record as viewed by the ProvideX application. The format consists of a series of field descriptors and/or literals, each separated by either a comma or a plus sign.

The simple format is:

**REC=** *fieldspec* { , | + } *fieldspec* ...

*Where:*

The *fieldspec* contains the name of the field and optional format, length, and scale. Fields are separated by either a comma or plus sign. When *comma-separated*, then a field delimiter is inserted. When *plus-separated*, then the field is padded to full size and no separator is inserted. Literals may be included if enclosed in apostrophes.

*Example:*

```
REC=CST_ID, NAME, ADDRESS
```

This results in a record with three fields, each separated by a field separator.

```
REC=CST_ID + NAME + ADDRESS
```

This results in a record consisting of three fields with each one padded to its full length and no intervening field separator. For example, if *CST\_ID* is 6 characters long and *NAME* and *ADDRESS* are both 30, then the record would be 67 characters long, including the record terminator.

Any column name can be followed by an optional colon and format specification. This format specification consists of a data type (if not numeric or string) followed by the field length. If the field is numeric, the length includes a decimal point followed by the number of decimal positions.

The possible data types are:

|   |                        |
|---|------------------------|
| P | Packed (BIN) data      |
| H | Data is stored in HEX  |
| B | Data is a Binary field |
| D | Field is a Date        |

Examples include:

```

CST_ID:7
OWING:8.2 (8 digits with 2 decimal places)
AMOUNT:P4.2 (4 bytes containing BIN value scaled by 100)
NAME:B30 (30 byte binary field)

```



It is a good idea to include the field descriptions for all fields since this prevents ProvideX from having to read the table's data dictionary to determine field sizes and types.

Hex and Binary values can be used to store non-printable and/or binary data that would cause problems otherwise when passed in a SQL statement.

Binary fields (type P) can be used to define numeric data that has been packed into a string using the **BIN( )** and **DEC( )** functions. If specified, the scale indicates the number of implied decimal places that the value contains.

Literals may be inserted within the record layout in order to insert padding where a field or column is not presently used, but space has been reserved for it. Literals should be enclosed with apostrophes and separated by a comma or plus sign.

### *Variant Record Processing*

In order to emulate multi-record type files (variant records) the database record must contain *all possible columns*; i.e., if record type 1 consists of the fields `Prefix` and `Value` when `Prefix="ABC"`, and record type 2 consists of the fields `Prefix` and `Percentage` when the 2<sup>nd</sup> and 3<sup>rd</sup> characters of `Prefix="EF"`, the database record would contain three columns `Prefix`, `Value` and `Percentage`.

**TYP=** specifies the field(s) that determine the record type. Using a **?** in the **REC=** clause defines the value.

Special masking options for **?** include:

- .** any one character (i.e., wildcard character).
- [abc]** any one of bracketed characters.
- [0-9]** any character from 0 to 9.
- [ ]** indicates end-of-field.
- ^** indicates records that don't match.

### *Example:*

```
TYP=Prefix;REC=?"ABC",Prefix,Value,?"EF",Prefix,Percentage
```

If the table contains two records:

```
"ABC",9,0
"AEF",0,99.99
```

Using the statement `READ (chan) A$, B:`

On the 1<sup>st</sup> **READ**, `A$="ABC"`, `B=9`.

On the 2<sup>nd</sup> **READ**, `A$="AEF"`, `B=99.99`.

`WRITE (chan) "XEF",50.5` would insert a new record into the database consisting of `"XEF",0,50.5`.



## [ODB] Tag

## Open DataBase

Format **OPEN** (*chan* [, *fileopt1* ]) "[ODB]*datasource* [ ; *table* ] [ ; *fileopt2* ]"

Where:

**[ODB]** File tag clause to inform ProvideX that it will be opening an external Windows ODBC database (not a ProvideX data file).

*chan* Channel or logical file number to open.

*datasource* Datasource name as defined in ODBC Administration.

*fileopt1* File options. Supported options include:  
**BSZ=***num* Buffer size (in bytes)  
**ERR=***stmtref* Error transfer  
**IOL=***iolref* Default IOList  
**NBF=***num* Dedicated number of buffers  
**OPT=***string* Open parameters (See [\[ODB\] OPT= Parameters](#))  
**REC=***string* Record prefix (**REC=VIS**(*string*) can also be used).

*fileopt2* **OPT=***string* Open parameters (See [\[ODB\] OPT= Parameters](#))

*table* Name of the table to open. If the table name is not supplied, then SQL statements sent to the database must be created by the application and sent via one of the following commands:

**WRITE** (*chan*) **SQL**\$

**WRITE RECORD** (*chan*) **SQL**\$

**READ** (*chan*, **KEY="!**, **SQL**\$)

See [DB2/ODB Table and Column Information, p. 775](#).



*Note:* The **[ODB]** tag is built into the ProvideX programming language. You do not need the ProvideX ODBC driver to use this tag, but you are limited to using the tag in Windows only. (ODBC is the Microsoft acronym for *Open Database Connectivity*.)

### Description

The **[ODB]** tag is used as a prefix in an **OPEN** statement to denote that ProvideX is to route all file I/O requests to an external ODBC database. Once you open a channel for **[ODB]** use, you can use it just like any other channel (i.e., for file I/O). It remains open until you close it.

ProvideX supports ODBC under Windows as well as two open source versions of ODBC for UNIX/Linux (*iODBC* and *unixODBC*). Use **TCB(197)** to determine if ODBC support is enabled for a given UNIX/Linux system.



*Note:* To open and read ProvideX (internal) data files using other database applications, (e.g., Excel), install and use the ProvideX ODBC driver instead of the **[ODB]** tag.

On the first use of an [ODB] tag under UNIX/Linux, ProvideX will first attempt to load the `unixODBC` shared library, `libodbc.so`. If that load fails, ProvideX will attempt to load the share library for ODBC, `libiodbc.so`. If that fails, an Error #15: Operating system command failed is reported.

### [ODB] OPT= Parameters

The **OPEN** parameters for connecting via ODBC are listed below: "

- "**ACCESS=**" Determines type of file access required (**READ** or **WRITE**). Default is **ACCESS=WRITE**. (*INI supported*)
- "**AUTOCOMMIT=**" Determines auto commit functionality of the database driver (either **ON** or **OFF**). (*INI supported*) It is applicable only if the driver supports transactions.
- "**CONCURRENCY=**" Determines the type of con-current access control/locking to be used. (*INI supported*) **READONLY** sets the cursor is set to read only - no updates allowed. **LOCK** applies low-level record locking. **OPT\_VERSION** causes optimistic locking with the database version control to be used. **OPT\_VALUE** causes optimistic locking with comparing record/column values to be used.
- "**COMPLETE=**" Determines the response to incomplete information by the following values:
- 0** (`SQL_DRIVER_NOPROMPT`). *Default*. Driver Manager copies the connection string specified by the application.
- 1** (`SQL_DRIVER_COMPLETE`) or **3** (`SQL_DRIVER_COMPLETE_REQUIRED`). If the connection string specified by the application includes the DSN keyword, the Driver Manager copies the connection string specified by the application. Otherwise, it takes the same actions as `SQL_DRIVER_PROMPT`.
- 2** (`SQL_DRIVER_PROMPT`). If connection string does not contain either `DRIVER`, `DSN`, or `FILEDSN` keyword, the Driver Manager displays the Data Sources dialog box. It constructs a connection string from the data source name returned by the dialog box and any other keywords passed to it by the application. If the data source name returned by the dialog box is empty, the Driver Manager specifies the keyword-value pair **DSN=Default**. (This dialog box will not display a data source with the name "Default".)
- All options except `NOPROMPT` require the handle of the parent window, which will be the handle of the currently active ProvideX window. This parameter is not used if a connection string is not supplied.
- "**CONNECT=**" Specifies a connection string surrounded by a delimiter character, enabling use of a "dsn-less" connection. Connection strings are driver specific. Consult the driver's reference for supported connection string values. Under UNIX/Linux, this parameter requires **COMPLETE=0** (see above).





If a connection string is supplied the value of the database name is ignored. If the database name is not null then the value is used only for determining if a connection should be shared.

For example,

```
Open(1,iol=*,opt="connect='DSN=nomads' ") "[odb]foo;Customer"
Open(2,iol=*,opt=" ") "[odb]foo;Customer Classes"
Open(3,iol=*,opt=" ",err=*next) "[odb];Customer"
Open(4,iol=*,opt="connect='DSN=foo' ") "[odb]foo;Customer"
```

The table opened on channel 2 will share the connection because of `foo`. Channel 3 will error-out because neither a valid database name nor a valid connect string was supplied. Channel 4 will open the table `Customer` using the properties of the "nomads" DSN because `foo` matches (the connect string was ignored).

If the keywords **USER=** or **PSWD=** are supplied on the open then the values of these attributes will be appended to the connection string.

- "CURSOR\_TYPE="** Defines the type of cursor that is to be used. (*INI supported*) **FORWARD** indicates that any result sets can be read in a forward only direction. **STATIC** indicates that the result set is static. **KEYSET** forces the cursor to use/maintain record keys in a Keyset. **DYNAMIC** indicates that the cursor is effective in the current Rowset only.
- "CURSOR\_USE="** Defines the type of cursor to be used within the ODBC connection. (*INI supported*) **DRIVER** (default) assumes the specific driver's own cursors. **ODBC** causes the ODBC interface to use the "Driver Managers" cursor library that may provide additional functionality not available within the database driver. **IF\_NEEDED** tells the system to use the specific database driver's own cursor functionality unless the additional functionality is requested specifically.
- "DATEFMT="** Date format mask applying to all date fields in table. (*INI supported*) This can be a combination of `YMD` with any other characters; e.g., to convert dates to 4-character year, month and day: `DATEFMT=YYYYMMDD`. Other characters are inserted as is; e.g., `DATEFMT=YY/MM/DD` with a date of March 1, 2004 would be returned as `04/03/01`.
- Two packed century formats are also supported. The first format, `AA`, maps A to 2000. Our example of March 1, 2004 with `DATEFMT=AAMMDD` would be returned as `A40301`. The second format, `KK`, is similar except `K` maps to 2000. A `DATEFMT=KKMMDD` would return `K40301` for the example.
- "DB="** or **"QUALIFIER="** Qualifies the specific database that you wish to use when using a driver to service multiple databases. (*INI supported*)

- "**DEBUGIT**=" String to append to SQL statement along with program name and line number for debugging purposes. (*INI supported*) This must indicate the comment character(s) appropriate to the database. For example, "--" is the comment identifier for Microsoft SQL Server; anything after "--" is ignored by SQL Server when compiling the SQL statement.
- "**EXEC\_SPRNO**=" Name of stored procedure used to emulate **RNO( )** function.
- "**EXTROPT**=" Controls the format of the **SELECT** statement used to process an **EXTRACT**. (*INI supported*) By default, **PVX** generates a **SELECT \* FROM table FOR UPDATE WHERE . . .**
- When **EXTROPT=text**, then *text* is substituted in place of **FOR UPDATE**. In addition, if the first character of *text* is \$, then the remaining characters of *text* are placed at the end of the **SELECT** statement rather than after the filename. This allows for different variations of **SQL** to be supported.
- "**FACTSDT**=" Causes all date fields to be translated to / from the **SQL** date format to the format used by the **FACTS** application.
- "**IND**=" Identifies a column that contains a sequential number starting at 0. This is used to emulate an indexed file.
- "**IGNORE\_NODATA**=" Set to **1 (Y or y)** to ignore **SQL\_NO\_DATA** error. If set to **0 (N or n)**, **ProvideX** will report Error #11: Record not found or Duplicate key on write if it receives a **SQL\_NO\_DATA** error when executing a direct **SQL** statement. Default is 1.
- "**ISOLATION**=" Controls the isolation that this connection will have relative to other processes on the same database. In particular, it controls Dirty reads (reading data that may be rolled back), Non-Repeatable reads (reading data after being changed by other transactions), and Phantom reads (reading data newly added to file). (*INI supported*)
- Settings include: **UNCOMMITTED** (*D, R, P* possible), **COMMITTED** (*D* possible, *R & P* not possible), **REPEATABLE** (*P* possible, *D & R* not possible), **SERIAL** (*D, R, & P* not possible).
- "**KEY**=" Identifies fields that make up the key(s). For named keys enter **\*NAME:keyname**; e.g.,
- OPEN(chan)"[ODB]dsn;table;KEY=field,field,\*NAME:keyname"**
- Use the **:D** option to indicate that the key segment is to be sorted in descending order; e.g., **KEY=KeyFld1,KeyFld2:D,KeyFld3**.
- "**KEYDATA**=" Identifies a column that represents the key. This is used to emulate an external key where the data is not duplicated in the data.
- "**KEYSET\_SIZE**=" Size of the Keyset for use with the cursor. (*INI supported*)

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "MAS90DATE"  | Reformats the contents of a date column to and from the Sage MAS 90 date format.                                                                                                                                                                                                                                                                                                                                                                                       |
| "MAS90SET"   | Sets flags for Sage MAS 90 emulation, such as turning on the <b>MAS90DATE</b> conversion.                                                                                                                                                                                                                                                                                                                                                                              |
| "MAXROWS="   | Maximum number of rows/records returned. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                      |
| "NONUMADJ="  | Set to <b>1</b> , <b>Y</b> or <b>y</b> to suppress +3 adjustment for defined length of numerics. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                              |
| "NONULLS="   | Inserts zero-length strings rather than nulls into the target database, and does not generate <b>WHERE</b> clauses checking for <b>IS NULL</b> or <b>IS NOT NULL</b> . ( <i>INI supported</i> ) Set to <b>1</b> , <b>Y</b> or <b>y</b> to enable or <b>0</b> , <b>N</b> or <b>n</b> to disable. If the application does not work correctly when moving from Version 5 or lower, then set <b>NONULLS=P</b> to indicate that keys are handled the same as pre-Version 6. |
| "NOSTRIP"    | Keeps trailing spaces (Default)                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| "NULLPADKEY" | Forces keys to be padded to full length with the null character, <b>\$00\$</b> . ( <i>INI supported</i> ) When used in INI file, set <b>NULLPADKEY=1</b> , <b>Y</b> or <b>y</b> .                                                                                                                                                                                                                                                                                      |
| "ORACLE="    | Indicates if the database uses ORACLE SQL sequence (either <b>Y</b> or <b>N</b> ). ( <i>INI supported</i> ) If <b>ORACLE=</b> and <b>TOP=</b> are used, then <b>SELECT</b> commands are generated as <b>SELECT * FROM (SELECT * FROM TABLE) WHERE ROWNUM &lt; 1</b> . Default is <b>ORACLE=N</b> .                                                                                                                                                                     |
| "POSUPDATE=" | Determines use of <i>SqlSetPos</i> functions. ( <i>INI supported</i> ) Use one of the following: <b>M</b> (must use positioned update), <b>O</b> (default, optionally use positioned update), <b>N</b> (never use positioned update).                                                                                                                                                                                                                                  |
| "PREPARE="   | Set to <b>1</b> , <b>Y</b> or <b>y</b> to use prepared statements. ( <i>INI supported</i> ) Prepared statements are pre-compiled SQL that may improve performance.                                                                                                                                                                                                                                                                                                     |
| "PSWD="      | Specifies password. ( <i>INI supported, but not secure</i> . Anyone with access to the INI will be able to read this password.)                                                                                                                                                                                                                                                                                                                                        |
| "REC="       | Provides the column names, type, and size. This is typically done to improve performance. If this information is not provided, then ProvideX must query the database for this information. For more information see <a href="#">ODB/OCI/DB2 Record Processing, p.789</a> .                                                                                                                                                                                             |
| "RECDATA="   | Identifies a column to return as the full record. This can be used for variant records which use complex rules to identify the record type.                                                                                                                                                                                                                                                                                                                            |
| "SHARED"     | Sets all tables to share a single connection to the Oracle database. Default.                                                                                                                                                                                                                                                                                                                                                                                          |
| "SCHEMA="    | Sets the Schema name to be prefixed to the table name (separated with a dot); e.g., <i>MySchema.MyTable</i> .                                                                                                                                                                                                                                                                                                                                                          |
| "STDDATE"    | Overrides the above formatting on individual columns.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| "STRIP"      | Removes trailing spaces from fields                                                                                                                                                                                                                                                                                                                                                                                                                                    |

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "TEXTMAX=" | Overrides maximum size for text fields (default is 4096 bytes).<br>( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| "TIMEOUT=" | Defines the time out value for any SQL operation (time before error 0 returned). ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| "TOP="     | Specifies use of the TOP clause in SELECT statements (limits the number of rows to return in a result set). ( <i>INI supported</i> ) If <b>TOP=n</b> is non-zero, then the <b>KEF()</b> / <b>KEL()</b> functions issue a <b>SELECT TOP 1 . . .</b> SQL statement, which improves system performance. If <b>TOP=n</b> > 0, then PVX issues <b>SELECT TOP n</b> to reduce the data transferred. <b>TOP=-1</b> indicates the driver supports <b>SELECT TOP</b> , but normal reading should not use it. Default is 0 ( <i>TOP not supported</i> ). |
| "TSQL="    | Defines a SQL statement that is used to control what data the logical file returns.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| "TYP="     | Sets identifier for different variant records. For more information see <a href="#">ODB/OCI/DB2 Record Processing, p.789</a> .                                                                                                                                                                                                                                                                                                                                                                                                                 |
| "UNIQUE"   | Sets new opens to be on a unique connection to the database. ( <i>INI supported</i> ) When used in an INI file, set <b>UNIQUE=1, Y</b> or <b>y</b> . <b>UNIQUE=0, N</b> or <b>n</b> indicates a shared connection.                                                                                                                                                                                                                                                                                                                             |
| "USER="    | Specifies login name. ( <i>INI supported</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

**Example**

To read Windows ODBC databases from a UNIX server, you would install and use WindX to make the connection or use ProvideX RPC on a Windows Server and open your ODBC databases through that server:

```
OPEN (14) "[WDX][ODB]datasourcename"
```

**See Also**

[READ Directive, p.271](#),  
[READ RECORD Directive, p.275](#),  
[SELECT..FROM..NEXT RECORD Directive, p.299](#),  
[WRITE Directive, p.383](#),  
[WRITE RECORD Directive, p.386](#)  
[OPEN Directive, p.232](#)



## [RPC] Tag

## Remote Process Control

|              |                                                                                                                                                                                                                                                                                                                                                                               |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Formats      | <ol style="list-style-type: none"> <li>1. <i>Call Remote Subprogram</i>: <b>CALL</b> "[RPC:server]subprog[;entry]"[,ERR=stmtref][,arglist],varlist</li> <li>2. <i>Open Remote File</i>: <b>OPEN</b> (chan[,fileopt])"[RPC:server]filename"</li> </ol>                                                                                                                         |
| Where:       |                                                                                                                                                                                                                                                                                                                                                                               |
| [RPC:server] | The RPC clause that initiates the remote <b>CALL</b> . <i>server</i> is the logical server name (see below).                                                                                                                                                                                                                                                                  |
| chan         | Channel or logical file number to open.                                                                                                                                                                                                                                                                                                                                       |
| ;entry       | Entry point label. (Optional.) If you include a label, precede it with a semicolon and append it to your subprogram name.                                                                                                                                                                                                                                                     |
| filename     | Name of the file to open prefixed by [RPC:server]. Use a string literal as in <code>OPEN (14) "[RPC:server]my_file"</code>                                                                                                                                                                                                                                                    |
| fileopt      | File options. Supported options include:<br><b>BSZ=num</b> Buffer size (in bytes)<br><b>ERR=stmtref</b> Error transfer<br><b>IOL=iolref</b> Default IOList<br><b>ISZ=num</b> Open file in binary mode<br><b>NBF=num</b> Dedicated number of buffers<br><b>OPT=char\$</b> File open options<br><b>REC=string\$</b> Record prefix ( <b>REC=VIS(string\$)</b> can also be used). |
| arglist      | One or more arguments (comma-separated if you include a list arg,arg...) to pass to the subprogram. Optional.                                                                                                                                                                                                                                                                 |
| server       | Name to be associated with a program server. Established using the <b>PROCESS SERVER Directive</b> , p.258.                                                                                                                                                                                                                                                                   |
| stmtref      | Program line number or label to transfer control to.                                                                                                                                                                                                                                                                                                                          |
| subprog      | Subprogram for the <b>CALL</b> directive; e.g.,<br><br><code>CALL "[RPC:INVENTORY]inv_alloc",a\$,b,c,d</code>                                                                                                                                                                                                                                                                 |
| varlist      | One or more variables, literals, mnemonics, <b>IOL=</b> options, and/or location functions; e.g., "@(5,4)". If you include a list, items must be comma-separated.                                                                                                                                                                                                             |

**Description** The [RPC] tag is used as a prefix to denote that ProvideX is to **CALL** a subprogram or open a file that resides on a remote server. Before a remote process control can be initiated, the server must be identified, and the *server* name established, via the **PROCESS SERVER Directive**, p.258.



*Note:* This feature requires ProvideX *RPC* activation. Refer to the ProvideX website for licensing information.

### Format 1: Call Remote Subprogram

**CALL** "[RPC:server]subprog[;entry]"[,ERR=stmtref][,arglist],varlist

The RPC **CALL** format tells ProvideX that your subprogram is to run (and that your data resides) on a different processor on the network; i.e., on a remote server. Use the same syntax as you would for a standard **CALL** directive, except that a **[RPC:server]** clause designates a remote server to handle the **CALL**. Since no data is transferred during RPC **CALL** processing, this helps maximize both network performance and data security. For more information, refer to the [CALL Directive, p.40](#).

#### Transparent RPC Process

At run time, ProvideX doesn't transfer data files or programs across the network for your RPC **CALL**. Instead, ProvideX puts your called remote subprogram name and parameters into a TCP/IP data packet and sends the packet to your remote server. The remote server loads and runs your subprogram, passing it your parameters. When your subprogram exits, the remote server puts your parameters (as altered by your subprogram) into a TCP/IP response packet and sends this back to the calling program.



*Note:* Since ProvideX RPC processing is true distributed processing, handled as a transparent background process, any displays during subprogram processing will appear on the remote server. Do not attempt to display subprogram activity on the calling machine.

### Format 2: Open Remote File

**OPEN** (chan[,fileopt])"[RPC:server]filename"

ProvideX supports the **OPEN** directive for remote files. Prefix your filename with the **[RPC:server]** clause to indicate that the server is to handle file requests.

#### See Also

[PROCESS SERVER Directive, p.258](#)

[OPEN Directive, p.232](#)

[Remote Process Capability Technical Overview.](#)

# [TCP] Tag Transmission Control Protocol

Format `OPEN (chan[,fileopt])"[TCP][server];socket[:tcp_opts]"`

Where:

**[TCP]** File tag that tells ProvideX the channel is being opened for a TCP/IP connection.

*chan* Channel or logical file number to open.

*fileopt* File options. Supported options include:  
**BSZ=num** Buffer size (in bytes, max. 32000).  
*Defaults: READ=1024, WRITE=32000.*  
**ERR=stmtref** Error transfer  
**TIM=num** Time-out value

*Example:* `OPEN ( chan , TIM=3 ) " [ TCP ] IP ; SOCKET "`

*arglist* One or more arguments (comma-separated if you include a list arg,arg...) to pass to the subprogram. Optional.

*server* Server address. Optional. If specified (to denote a client connection on the channel), use either an IP address like 172.16.1.1 or a DNS (Domain Name System) server, such as *www.pvx.com*. Omit it to indicate a host connection.

*socket* TCP/IP socket number. For a host connection, this is the socket number where the host listens and the client connects. For a client connection, this is the socket number where the host listens. The valid range for [TCP] sockets is 1 to 65535. (For an invalid socket number, less than 0 or greater than 65535, the OS dynamically assigns an unused valid number.)

For more information, see [System Limits, p.825](#)

*tcp\_opts* Options to override default TCP characteristics. When including a list, use the semicolon as a separator. Supported options are as follows:

**BINDTO=** *For server-side.* Restricts service requests to a specified address. By default ProvideX binds a server style socket that is open to all IP addresses in a machine. This sets your program to only monitor the socket number on a specific IP address, allowing other software to use the same socket on different IP addresses.

**KEEPAIVE** *For client-side.* Forces the OS to send *keepalive* packets to the host, thereby keeping TCP/IP pipes open forever on TCP/IP connections which time out due to inactivity; e.g., `OPEN ( chan ) " [ tcp ] ip ; socket ; KEEPAIVE`

**NODELAY** Disables algorithm that delays transmission. *Default:* delay up to 200ms., attempt to combine data into larger packets.

**REUSE** Allows server to monitor a port that may currently be in use.

**SECURE** *For client-side.* Uses the certificate found by the host to negotiate and carry out encrypted communication.

**SECURE=** *For server-side.* Sets path to public certificate used for encoding and decoding communications between client and server.

`SINGLE` For server-side. One client per connection.  
*Default:* multiple clients (*not single*).

`STREAM` Sets streaming data mode. *Default:* block mode.

**Description** The **[TCP]** tag is used as a prefix in an **OPEN** statement to denote that ProvideX is to open the channel for a TCP/IP connection.

The TCP interface in ProvideX acts like a "smart" two-way communications pipe. A ProvideX TCP connection can be either a Server or Client-style connection; these are explained in the following sections. A single program can have many TCP sockets opened concurrently, with each independent of the others. However, there are operating system limitations. For more information, see [System Limits, p.825](#).

### *TCP Server Connection*

Omit the *server* value from the **TCP OPEN** syntax to notify ProvideX that the channel is to be opened as a host/server.

#### *Example:*

```
OPEN (1,BSZ=8192) "[TCP];10000"
```

This opens a TCP channel as a host/server connection with a block size of 8192 bytes. A host/server can communicate with more than one client at a time. (You can override this by using `SINGLE` as one of your *tcp\_opts*.)

### *TCP Client Connection*

Include the optional *server*, to notify ProvideX that the channel is to be opened as client connection linked to the socket where the host is listening; i.e., this opens a channel to create a link to the host's socket number. A client can only communicate with the specific host to which it's connected.

#### *Examples:*

```
OPEN (1,BSZ=16384) "[TCP]172.16.1.1;10000"
```

The example above opens a client link to a TCP channel, port 10000 on the server. The client PC uses any available socket number for its side of the communications. Note that the local machine dynamically assigns an available socket for your program to use.

The [KEF\(\) Function, p.466](#), can be used to identify local sockets; e.g.,

```
LET MY_KEY$=KEF(CHAN)
```

The data in `MY_KEY$` is in IP format, "10.12.1.12;80;GORDD".

As another example, `open (1) "[tcp]www.microsoft.com;80;nodelay"` opens a connection to Microsoft's website, which is listening to socket number 80 (the default). This also turns off the 200ms delay packet optimization algorithm.

**See Also**

[OPEN Directive, p.232](#)  
[ProvideX Client-Server Reference](#)





# [WDX] Tag Direct Action to Client Machine

## Formats

1. *Execute*: EXECUTE "[WDX]statement"
2. *Invoke*: INVOKE "[WDX]statement"
3. *Call Subprogram*: CALL "[WDX]subprog",params
4. *Open File*: OPEN (chan[,fileopt])"[WDX]filename"
5. *Open COM Port*: OPEN (chan,OPT=settings\$)"[WDX]port\_id"
6. *Open Tag Process*: OPEN (chan[,fileopt])"[WDX][tag]prog;[params]"
7. *OPEN Print Devices*: OPEN [INPUT] (chan[,fileopt])"[WDX]\*device\*[:Q\_name[:Q\_options]]"
8. *Create Object*: NEW("[WDX]ClassName")
9. *Define Windows Object*: DEF OBJECT com\_id,"[WDX]objname"
10. *Call \*WindX.utl*: CALL "[WDX]\*WindX.utl;function",params

## Where:

|           |                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [WDX]     | File tag notifies ProvideX that you are directing the action to the WindX client machine instead of the server.                                                                                                                                                                                                                                                               |
| chan      | Channel or logical file number to open.                                                                                                                                                                                                                                                                                                                                       |
| class\$   | Name of class for creating new object. String expression.                                                                                                                                                                                                                                                                                                                     |
| com_id    | Numeric variable to receive a handle (memory pointer to object).                                                                                                                                                                                                                                                                                                              |
| *device*  | Identifier and parameters for either of the two [WDX]-supported special device files (*WINDEV* or *WINPRT*). See <a href="#">*WINDEV* Raw Print Mode, p. 756</a> , and <a href="#">*WINPRT* Windows Printing, p. 760</a>                                                                                                                                                      |
| Q_name    |                                                                                                                                                                                                                                                                                                                                                                               |
| Q_options |                                                                                                                                                                                                                                                                                                                                                                               |
| filename  | Name of the file to open (file must exist on the client PC); e.g.,<br>OPEN (14) "[WDX]+TMP\$ or<br>OPEN (14) "[WDX]temp_file"                                                                                                                                                                                                                                                 |
| function  | A ProvideX utility/function, part of the *WindX.utl utility program. For instance, the *WindX.utl;Spawn function initiates tasks on the server and/or client.                                                                                                                                                                                                                 |
| fileopt   | File options. Supported options include:<br><b>BSZ=num</b> Buffer size (in bytes)<br><b>ERR=stmtref</b> Error transfer<br><b>IOL=iolref</b> Default IOList<br><b>ISZ=num</b> Open file in binary mode<br><b>NBF=num</b> Dedicated number of buffers<br><b>OPT=char\$</b> File open options<br><b>REC=string\$</b> Record prefix ( <b>REC=VIS(string\$)</b> can also be used). |
| objname\$ | Name by which the COM object is registered in the Windows system registry subkey HKEY_CLASSES_ROOT.                                                                                                                                                                                                                                                                           |
| params    | Arguments and variables you pass to the subprogram or function. If you include a list, it's comma-separated.                                                                                                                                                                                                                                                                  |
| port_id   | System identifier for the port; e.g., COM1.                                                                                                                                                                                                                                                                                                                                   |

|                      |                                                                                                                                                                                                                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>prog;function</i> | Parameters for the [tag]. For instance, for [DDE] to export to Excel, you can open a worksheet using either,<br>" [WDX] [DDE] EXCEL;existing_worksheet.WK1 " <i>or</i><br>" [WDX] [DDE] EXCEL; "<br>In the first example, where a spreadsheet name is included, it must exist on the client PC. |
| <i>tag</i>           | ProvideX special command file tag. (All are listed in this chapter); e.g., [DDE], [DLL], etc.                                                                                                                                                                                                   |
| <i>settings\$</i>    | Serial device's attributes (baud rate, etc.). String expression.                                                                                                                                                                                                                                |
| <i>statement</i>     | Command supported by ProvideX ( <i>some are not</i> ) for use with <b>EXECUTE</b> or <b>INVOKE</b> + the [WDX] tag in WindX. String expression.                                                                                                                                                 |
| <i>subprog</i>       | Subprogram for the <b>CALL</b> directive.                                                                                                                                                                                                                                                       |



*Note:* This feature requires a *WindX* activation. Refer to the ProvideX website for licensing information.

## Description

The [WDX] tag is used as a prefix to perform an action on a remote *client* machine (*if* your given command could be used on either client or server, and *if* the command supports using [WDX]). The [WDX] tag can only be used in specialty commands when running under *WindX* in a client-server environment.

WindX supports the use of the following commands via the [WDX] tag: **SERIAL**, **KEYED**, **DIRECT**, **SORT**, **PROGRAM**, **DIRECTORY**, **REFILE**, **LOCK**, **UNLOCK**, **MNEMONIC** and **ERASE**.



*Note:* If you are running an earlier version of ProvideX on a WindX PC, you must encapsulate these commands in an EXECUTE " [WDX] . . . " or design your applications to run on the remote site. *Also*, the **DIR=** option and the **PURGE**, **FILE** and **INDEXED** directives are not supported by WindX and must be encapsulated in an **EXECUTE** etc.

### To Detect WindX

To detect whether or not you're actually working on a WindX PC in Command mode, look for the special prompt **-}** ... i.e., a hyphen with curly brace. In your applications, a test for one of the following indicators will tell you that your session is running under WindX:

```
DEC (MID (MSE , 22 , 1)) > 0
TCB (88) < > 0
```

For more information, refer to the **TCB ( ) Function, p.534**, and the **MSE System Variable, p.565**.

You can create a global variable in your WindX setup, and use it later to test for the presence of a WindX client PC connection; then, if WindX is running the session, you can use the [WDX] tag. In the following example, the user-defined variable %WDX below is only set when WindX is running, otherwise its value is null:

```
IF MID (MSE , 22 , 1) > 00 AND MID (MSE , 22 , 1) < FF THEN %WDX$ = " [WDX] "
```



Use such a global variable in a statement like `OPEN ( 30 )%WDX$+ " *WINPRT* "` to bypass an Error #12: File does not exist (or already exists) when WindX isn't running.

### *How ProvideX Detects WindX*

ProvideX uses terminal type to detect a WindX session. In UNIX, ProvideX recognizes two terminal types as potential WindX stations (`TERM="winterm"` for the WindX client PC and `TERM="ansi"`). Since most UNIX systems and applications can't recognize or use the `winterm` type, the ProvideX `ansi` device driver sends a special escape sequence to a terminal to test for a WindX client. If the terminal is a WindX PC, a special response is generated. If no response is generated before a time-out occurs, ProvideX assumes the device is `ansi`.

### *GUI Requests*

Once the ProvideX session on the server recognizes the terminal as a WindX station, it changes the internal settings to allow graphical requests to be routed correctly. (Then, graphical requests are automatically tokenized and forwarded to WindX for processing. That is, you do not need the [WDX] tag.)

For instance, a server command to print a picture in a UNIX environment would automatically be sent to WindX for the client. ProvideX transmits standard mnemonics to WindX as an escape (`$1B$`) followed by the mnemonic in native form. Traffic from the host/server is minimized because the WindX client's ProvideX interpreter handles a lot of the functionality locally (on the client) for screen refreshing and graphical requests.

In WindX, your instruction to print is sent to the client, bundled as is. When you use mnemonics and/or graphics like `.bmp`'s, they must exist on the client or be accessible to the client. The following example uses a '**PICTURE**' that is defined and shared on a common Windows server instead of being stored on each individual client machine:

```
print 'picture' (10,10,10,10), "\\serv_name\driveshare\your_bmps\that.bmp"
```

See Also *ProvideX Client-Server Reference*

## Formats 1 & 2: *Initiate Remote Command*

**EXECUTE** "[WDX]statement"

**INVOKE** "[WDX]statement"

Use the **EXECUTE** and **INVOKE** formats with the [WDX] tag to process commands on the WindX PC (remote client). Common applications of the **EXECUTE** format would be: changing the client's local directory, setting system parameters, or altering the prefix and in file creation.



**Warning:** When you use the **EXECUTE** and **INVOKE** directives from your server to initiate action remotely on a client, the client PC might be running a ProvideX activation with a different set of syntax tables. As a result, your **MNEMONICS** might be invalid.

The following example illustrates the use of an **EXECUTE "[WDX]statement"** in an application on the host/server to define a **MNEMONIC** directive locally on the WindX client:

```
IF WDX%<->00 THEN EXECUTE "[WDX]MNEMONIC(LFO)'5X'=$1B+hex$" ELSE GOTOMY_LABEL
```

The following example sets the **'B0' System Parameter**, [p.656](#), on a WindX PC using a **[WDX]** command from a UNIX host:

```
EXECUTE "[wdx]SET_PARAM 'B0' "
```



**Note:** If you run your application with **'B0'** set, make sure that it is set on both the host and the WindX PC. Otherwise, the wrong windows will be addressed. (You can set **'B0'** either by executing it from the host, as in the example, or by using **-B0** as an argument on the WindX PC's startup command line.

### Format 3: Call Subprogram

```
CALL "[WDX]subprog",params
```

This format enables the server to **CALL** a subprogram that exists and runs remotely on the client. It is true distributed processing. The server's files and global variables are not accessible to your subprogram. Your remote call passes parameters back and forth, not data files. You can pass a maximum of 20 parameters/arguments, in a comma-separated list.

For example, you can call applications like special printer device drivers you have built on the client to handle print mnemonics:

```
CALL "[WDX]*dev/your_driver_name"
```

You can use a similar format to call the ProvideX `*windx.utl` (utility) functions. For further information, see [Format 10: \[WDX\] and \\*WindX.utl, p.806](#), [\[WDX\]\\*WINDEV\\* Escape Sequences, p.758](#).

### Format 4 & 5: Open Files, Serial Ports on Client

```
OPEN (chan[,fileopt])"[WDX]filename"
```

```
OPEN (chan,OPT=settings$)"[WDX]port_id"
```

Use this format in programs running on your server when you're opening remote files or ports. That is, using **[WDX]** as a prefix to the pathname of your file (or port, etc.) tells WindX to open it remotely on the client. WindX automatically forwards requests (i.e., file I/O directives) for processing.



**Reminder:** When you need a statement that is not supported on a WindX PC, encapsulate your commands in an **EXECUTE "[WDX] . . . "** directive or design your applications to run remotely on the client.

**Examples:**

The following example opens a remote file,

```
OPEN (1) "[WDX]CLIENT_PATH\CLIENT_FILE"
```

Use a similar format to open a COM port for direct access to a serial device (e.g., a serial printer or weigh scale) on the client PC without going through a spooler,

```
LET SETTING$="9600,n,8,1,x"
OPEN (5,OPT=SETTING$) "[WDX]COM2"
or
OPEN (5,OPT="9600,n,8,1,x",ERR=BADCOM) "[WDX]COM2"
```

### Format 6: [WDX] with Other Tags

```
OPEN (chan[,fileopt])"[WDX][tag]prog;[params]"
```

[WDX] can be used in conjunction with other file tags; e.g.,

```
0010 OPEN (1) "[WDX][DDE]excel;existing_worksheet.wk1"
```

### Format 7: Windows Print Subsystem on Client

```
OPEN [INPUT] (chan[,fileopt])"[WDX]*device*[:Q_name[:Q_options]]"
```

ProvideX opens the special device files **\*WINPRT\*** and **\*WINDEV\*** on the PC/server which issues the command. Except in UNIX (where it's done automatically), use [WDX] with **OPEN [INPUT]** directives for the two specialty files to direct any print jobs and dialogues to the WindX client PC, which will in turn use its Windows print subsystem API to deal with the jobs and send them to the given printer; e.g.,

```
OPEN (7,ERR=1500) "[WDX]*WINPRT*"
```

With a WindX client and an Windows Server, if you *do not* use [WDX] in your **OPEN** directive, then the printer selection dialogue will appear on the server console, and any print queue you name directly must exist on the Windows Server in the Control Panel printers folder.

See also [\[WDX\]\\*WINDEV\\* Escape Sequences, p.758](#), [\\*WINDEV\\* Raw Print Mode, p.756](#), and [\\*WINPRT\\* Windows Printing, p.760](#).

### Formats 8 & 9: Remote Object Support

```
NEW("[WDX]ClassName")
DEF OBJECT com_id,"[WDX]objname"
```

The [WDX] tag can be used to create OOP/COM objects and manipulate them across a WindX connection as if they existed locally.

Methods can be passed arguments and receive results; however, only the result of a method will be returned across a remote connection. Any changes to the arguments of a method by the remote object will not be returned across the connection.

Arguments are therefore considered to be passed by value only. If you need to retrieve arguments as well as the result, you must place your code in a program on the WindX workstation and interact with that code.

Event handling is not supported across a remote connection. Event mapping must occur within the remote object. The remote object will not have access to any server resources. At most, the remote event could pass a CTL back to the local server for action. It is recommended that objects requiring event processing exist completely on the remote and interact only with the local WindX session on the remote.

### Format 10: [WDX] and \*WindX.utl

**CALL "[WDX]\*WindX.utl;function",params**

Use this format to call functions in the WindX.utl utility program; e.g.,

```
call "[wdx]*windx.utl;Get_LWD",Station_dir$
call "*windx.utl;Spawn",cmdline$,infile$,appid$
```

The functions supplied by this utility are listed and described below:

|                                                          |                                                                                                                                                                                                                  |
|----------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "[WDX]*WindX.utl;CWDIR",D\$                              | Changes to the specified directory D\$ on the WindX client.                                                                                                                                                      |
| "[WDX]*WindX.utl;Get_Addr",X\$                           | Returns the IP address of the WindX client.                                                                                                                                                                      |
| "[WDX]*WindX.utl;Get_ARG",X,Y\$                          | Returns the command line argument number specified by X in Y\$.                                                                                                                                                  |
| "[WDX]*WindX.utl;GET_DSK",X,Y\$                          | Returns the <b>DSK()</b> information for the disk specified by X (or X\$) in Y\$.                                                                                                                                |
| "[WDX]*WindX.utl;GET_DIR",X,Y\$                          | Returns the <b>DIR()</b> information for the directory specified by X (or X\$) in Y\$.                                                                                                                           |
| "[WDX]*WindX.utl;GET_FILE_BOX",P\$,D\$,W\$,E\$,L\$       | Emulates a local call to <b>GET_FILE_BOX</b> directly on the WindX client. For complete information, see <a href="#">GET_FILE_BOX, p. 139</a> .                                                                  |
| "[WDX]*WindX.utl;GET_FILE_BOX_READ",P\$,D\$,W\$,E\$,L\$  |                                                                                                                                                                                                                  |
| "[WDX]*WindX.utl;GET_FILE_BOX_WRITE",P\$,D\$,W\$,E\$,L\$ |                                                                                                                                                                                                                  |
| "[WDX]*WindX.utl;GET_FILE_BOX_DIRECTORY",P\$,D\$,W\$,R\$ |                                                                                                                                                                                                                  |
| "[WDX]*WindX.utl;Get_LPG",X\$                            |                                                                                                                                                                                                                  |
| "[WDX]*WindX.utl;Get_LWD",X\$                            | Returns the local current disk directory <b>LWD</b> for the WindX session.                                                                                                                                       |
| CALL "*WindX.utl;Spawn",X\$,I\$,F\$                      | <b>No [WDX] prefix required.</b> Spawns a new session of ProvideX on the host and an associated WindX session on the client PC. By default, if the main session terminates, then the spawned session terminates. |

The parameters are:

X\$ command line parameters to be used on the host.

I\$ pathname of INI file to be used on the client PC.

F\$ value of FID ( 0 ) for the session.

|                                                          |                                                                                                                                                                                  |
|----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CALL " <code>*WindX.utl;Spawn_Nohup</code> ",X\$,I\$,F\$ | <b>No</b> [WDX] <i>prefix required</i> . Same as above but detaches the session from the main user task. If the main task terminates, then the spawned task continues executing. |
| CALL "[WDX]*WindX.utl;Get_WindX",X\$                     | Returns the absolute pathname of the WindX program.                                                                                                                              |
| CALL "[WDX]*WindX.utl;Get_NewPort",X                     | Returns the port number of an unused TCP/IP port on the WindX station.                                                                                                           |
| CALL "[WDX]*WindX.utl;Get_TCB",X                         | Returns the value of the <b>TCB( )</b> function task specified by X.                                                                                                             |
| CALL "[WDX]*WindX.utl;Get_Val",X\$,Y\$                   | Evaluates/returns value of string expression X\$ in Y\$; i.e., Y\$=EVS(X\$).                                                                                                     |
| CALL "[WDX]*WindX.utl;Get_Num",X\$,Y                     | Evaluates/returns value of numeric expression X\$ in Y; i.e., Y=EVN(X\$).                                                                                                        |



**Note:** Calls to `*WindX.utl;spawn` and `*WindX.utl;Spawn_Nohup` do not require the [WDX] prefix because they are performed locally.







# A

## Appendix

---

### Overview

This appendix discusses additional features of the ProvideX language, such as file options, labels, operators, and control codes. It also includes supplementary information regarding preset system limits, error messages, and reserved words. Section headings, page numbers, and outlines are listed below:

**Input/Output and Control Options, p.810**

Various options that can be included in a directive or system function to fine tune code and redirect processing.

**Data Format Masks, p.813**

Character string masks used to define data (input/output) in ProvideX.

**Labels/Logical Statement References, p.816**

Built-in line labels that can be used instead of line number references in applications.

**Negative CTL Definitions, p.817**

Actions that are assigned to negative CTL values, and have special significance to ProvideX.

**Operators, p.821**

All operators that can be used in ProvideX, including: property, assignment, auto-increment/decrement, **LIKE**, and the **Apostrophe Operator, p.823**.

**System Limits, p.825**

Preset system limits of ProvideX.

**Reserved Words, p.827**

Words that are reserved for internal use by ProvideX.

**Error Codes and Messages, p.828**

The numeric system variable **ERR**, error codes, and the messages associated with errors detected by ProvideX.



# Input/Output and Control Options

Several directives and system functions described in this document include the use of optional syntax elements in their formats to fine tune code and redirect processing. Some are defined individually, and some are listed in format groups: **Control Options** (*ctrlopt*) or **File Options** (*fileopt*). A general overview of available options is provided below. Refer to specific directive/function descriptions for information on the use of these options.

## File Options

File options (*fileopt*) can be included in the syntax for **Accessing Data Files**, p.22. These may be used to handle exceptions, set key position, and deal with data errors in I/O operations:

|                     |                                                                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>BSY=stmtref</b>  | To trap Error #0: Record/file busy.                                                                                                                                                                                                                                 |
| <b>BSZ=num</b>      | <i>Buffer size</i> (in bytes) for file opens and directives that control data flow. <i>Block size</i> (in KB) for file creation directives ( <b>CREATE TABLE</b> , <b>KEYED</b> , etc.).                                                                            |
| <b>DIR=num</b>      | Direction indicator. This adjusts the record pointer by <i>num</i> records, where a positive value advances the pointer, a negative pointer reverses the pointer, and a DIR=0 indicates no movement. <i>This option is not supported with use of the [WDX] tag.</i> |
| <b>DOM=stmtref</b>  | On missing record, transfer to program line number/line label.                                                                                                                                                                                                      |
| <b>END=stmtref</b>  | On end of file, transfer to program line number/line label.                                                                                                                                                                                                         |
| <b>ERR=stmtref</b>  | On error, transfer to program line number/line label.                                                                                                                                                                                                               |
| <b>HLP=string\$</b> | Help message identifier used with <b>INPUT</b> and <b>OBTAIN</b> .                                                                                                                                                                                                  |
| <b>IND=num</b>      | 32-bit record index value used to uniquely identify a record in keyed files.                                                                                                                                                                                        |

For *Fixed length keyed* files, *num* represents an offset into the data file (first record has an index of 0, second is 1, and so on). However, some record indexes will be set aside by the system to be used for key tables and may yield gaps where the record indexes have been used for keys.

For *Variable length keyed* files, *num* represents a logical page address and record index within that page. The page address is contained in the top 24-bits (high order 3 bytes) with a record index within that page in the lower 8 bits. For VLR files, the page address is the actual physical address for the data page. For EFF files, the page address is a logical page number in the file.

For *TCP/IP server* files, *num* represents an internal socket connection to the client that can be used to manually direct output to specific sockets.

Used with the **INPUT** directive, **IND=num** sets the starting position (column number) of the cursor in the input field.

|                                          |                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IOL</b> = <i>iolref</i>               | Either a string variable containing the object code of an IOList or a statement reference to an IOList (statement number/label).                                                                                                                                                                                                                             |
| <b>ISZ</b> = <i>num</i>                  | File open for access in binary mode.                                                                                                                                                                                                                                                                                                                         |
| <b>KEY</b> = <i>string</i> \$            | <i>Record key</i> or <i>Password to open file</i> .                                                                                                                                                                                                                                                                                                          |
| <b>KNO</b> = <i>num</i>   <i>name</i> \$ | File access key number ( <i>num</i> ) or name ( <i>name</i> \$), where <i>num</i> is 0 based (0-15 for VLR/FLR files, 0-255 for EFF files).                                                                                                                                                                                                                  |
| <b>NBF</b> = <i>num</i>                  | Dedicated number of buffers.                                                                                                                                                                                                                                                                                                                                 |
| <b>NUL</b> = <i>stmtref</i>              | On no input, transfer to program line number/line label.                                                                                                                                                                                                                                                                                                     |
| <b>OPT</b> = <i>string</i> \$            | File open options.                                                                                                                                                                                                                                                                                                                                           |
| <b>REC</b> = <i>name</i> \$              | Record prefix represented by actual <i>name</i> , not the contents of <i>name</i> \$. Use <b>REC=VIS(<i>string</i>\$)</b> to obtain the name from inside a variable.                                                                                                                                                                                         |
| <b>RNO</b> = <i>num</i>                  | Record number.                                                                                                                                                                                                                                                                                                                                               |
| <b>RTY</b> = <i>num</i>                  | Number of times to retry (one second intervals). Default is set via the ' <b>WT</b> '= <a href="#">System Parameter, p.694</a> .                                                                                                                                                                                                                             |
| <b>SEP</b> = <i>char</i> \$              | Default field separator character. Hex or ASCII string value.                                                                                                                                                                                                                                                                                                |
| <b>SIZ</b> = <i>num</i>                  | <i>Number of characters to read</i> : If negative, <i>num</i> identifies the number of characters to be read. If <i>num</i> is a positive number, the read continues until <i>num</i> characters are received.<br><i>Number of bytes to write</i> : If <i>num</i> exceeds the amount of data being written from the variable, the data is padded with nulls. |
| <b>TBL</b> = <i>stmtref</i>              | Data translation table.                                                                                                                                                                                                                                                                                                                                      |
| <b>TIM</b> = <i>num</i>                  | Maximum time delay to wait.                                                                                                                                                                                                                                                                                                                                  |

## Control Options

Control options (*ctrlopt*) can be included in the syntax for creating and maintaining various [Graphical Control Objects, p.21](#).

|                                            |                                                                                                                                  |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>ERR</b> = <i>stmtref</i>                | On error, transfer to program line number/line label.                                                                            |
| <b>HLP</b> = <i>string</i> \$              | Help message identifier for defining <a href="#">AutoComplete, p.218</a> .                                                       |
| <b>FNT</b> =" <i>font,size[,attr]</i> "    | Font name, size, optional attributes Refer to the ' <b>FONT Mnemonic, p.609</b> ', for details.                                  |
| <b>FMT</b> = <i>def</i> \$  <i>mask</i> \$ | Format definition for the associated control. If used for character string masks, see <a href="#">Data Format Masks, p.813</a> . |
| <b>KEY</b> = <i>char</i> \$                | Hot key                                                                                                                          |
| <b>LEN</b> = <i>num</i>                    | Maximum input characters.                                                                                                        |
| <b>MSG</b> = <i>text</i> \$                | Message line.                                                                                                                    |
| <b>MNU</b> = <i>ctl</i>                    | CTL value associated with right-click menu event.                                                                                |

|                      |                                                                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NUL</b> =string\$ | Empty value.                                                                                                                                                                                                                                                       |
| <b>OWN</b> =name\$   | Name assigned to a control for automated testing purposes. This will be visible to programs that use the Microsoft Active Accessibility (MSAA) interface. For detailed information on this subject, refer to the document <i>ProvideX GUI Testing Automation</i> . |
| <b>OPT</b> =char\$   | Single character attribute/behaviour settings. Settings may be combined. Invalid settings are ignored.                                                                                                                                                             |
| <b>SEP</b> =char\$   | Column delimiter. Hex or ASCII string value.                                                                                                                                                                                                                       |
| <b>TBL</b> =char\$   | Single character translation.                                                                                                                                                                                                                                      |
| <b>TIP</b> =text\$   | Mouse pointer message. Refer to the <b>'TC'= System Parameter, p.688</b> , to change the colour.                                                                                                                                                                   |

# Data Format Masks

A format mask is a character string that can be used to define how data is to be displayed or printed in ProvideX (**PRINT Directive**, p.255). Masks can also be applied to filter data being received from the keyboard (**INPUT Directive**, p.160) or in the conversion/validation of a string (**STR( ) Function**, p.525).

For further information on data input and output in ProvideX, refer to the *ProvideX User's Guide*.

## Assigning a Format Mask

To assign a format mask in ProvideX syntax, place a colon before the mask following the given data value:

```
val[$]:mask$
```

*mask\$* may be a literal string, a string variable, substring, or a string expression (concatenation); e.g.,

```
0010 PRINT "The total is ",A:"$#,###,##0.00CR"
```

or

```
0010 LET MASK$="000-0000"
```

```
0020 PRINT "Phone: ",T:MASK$
```

The number of characters defined in a mask must be equal to or larger than the number of characters to be displayed. One output character is generated for each character present in the format mask.

When more characters exist in the data value than are specified in the format mask, the result will generate an **Error #43: Format mask invalid**; e.g., outputting 1000 with a mask of "##0" causes an error. However, the parameters **'FI'**, **'F'** and **'FO'** can be specified to handle overflows without generating errors.

## Format Defaults

If no format mask is specified when outputting numeric values, the system formats the value as follows:

- The first character output will indicate the sign of the value. A space will be output if positive, a minus sign if negative.
- If the absolute value is greater than 10E+18 or is less than 10E-18 (but not zero), the value is output using scientific notation.
- The number is rounded to the current precision in effect and output suppressing all leading zeroes and all trailing zeroes following the decimal point. The decimal point is suppressed if no digits remain after it.

For example, assuming precision of two:

```
->0010 PRINT 3/2, 6/3, 3-4, 2/3
1.5 2-1 .67
```

## Numeric Format Masks

Numeric format mask characters are used to convert numeric data (from literals, variables, or numeric expressions) to ASCII. Format masks allow for the generation of fixed format data with the insertion of fill characters (usually a space) to suppress leading/trailing zeroes.

The recognized numeric format mask characters are described below:

| <i>Character</i> | <i>Description</i>                                                                                                                                                                                                                                       |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0                | <i>Zero</i> . Outputs one digit from the numeric value.                                                                                                                                                                                                  |
| #                | Outputs one digit of the number unless the digit is zero and no digits have been output yet, in which case it outputs a fill character. (Suppresses leading zeroes).                                                                                     |
| .                | Outputs/aligns decimal point. One occurrence allowed per mask.                                                                                                                                                                                           |
| !                | Treated as '.' but causes output to be replaced with spaces if the value is 0 <i>zero</i> ; i.e., 'Blank when Zero'.                                                                                                                                     |
| -                | In front end of a mask, inserts '-' (if value negative) or a fill character (if positive) just before the first digit. In the trailing end of a mask, outputs either a '-' or a fill character. Within the mask, outputs a dash regardless of the value. |
| +                | Outputs either a '-' if the number is < 0 otherwise outputs a '+'. If this format mask character is in front of the number, the output is placed just before the first digit. (floating +)                                                               |
| ,                | Outputs a comma if some digits have been output; otherwise it outputs a fill character.                                                                                                                                                                  |
| \$               | If at the front of the number, indicates that a dollar sign is to be output in front of the first digit of the number (floating dollar sign). Anywhere else, indicates that a dollar sign is to be output.                                               |
| *                | If before any digits of the number, causes asterisks to be used as the fill character instead of a space. If it occurs anywhere else within the mask, it causes an asterisk to be output.                                                                |
| ( or )           | Outputs parentheses if the value is negative; otherwise it outputs a fill character.                                                                                                                                                                     |
| CR               | Outputs CR if the value is negative; otherwise it outputs two fill characters.                                                                                                                                                                           |
| DR               | Outputs CR if the value is negative; otherwise it outputs DR.                                                                                                                                                                                            |
| B                | Outputs a space.                                                                                                                                                                                                                                         |
| <i>other</i>     | Outputs the character.                                                                                                                                                                                                                                   |

Before being output, the number is rounded to the number of decimal places specified in the format mask. If no sign indication is specified (i.e., no -, +, (, ), CR, or DR in mask), no sign will be output.

The following table shows the results of various masks used on different values:

| <i>Value</i> | <i>Mask</i>     | <i>Result</i>      |
|--------------|-----------------|--------------------|
| 1            | "000000"        | "000001"           |
| 1            | "####0"         | " 1"               |
| -2.4         | "-###0.00"      | " -2.400"          |
| 1000.9       | "#,##0+"        | "1,001+"           |
| -10.5        | "\$#,##0.00BDR" | " \$10.50 CR"      |
| 5551212      | "000-0000"      | "555-1212" (Phone) |
| 2359         | "00:00"         | "23:59" (Time)     |
| -45          | "####0"         | " 45"              |

## String Format Masks

String data can also be converted through the use of format masks. Unlike numeric format masks, string format masks are typically used to validate that the contents of a string match a pre-defined format.

The recognized string format mask characters are described below:

| <i>Character</i> | <i>Description</i>                                                                                                                                                    |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0                | <b>Zero.</b> String must contain either a digit (0-9) in this position.                                                                                               |
| A                | String must contain an alphabetic letter (A-Z, a-z) in this position. The output of the format mask is converted to uppercase.                                        |
| a                | String must contain an alphabetic letter (A-Z, a-z) in this position.                                                                                                 |
| X                | String can contain any character in this position. The output of the format mask is converted to Upper case.                                                          |
| x                | String can contain any character in this position.                                                                                                                    |
| Z                | String must contain an alphabetic letter (A-Z, a-z) or a digit (0-9) in this position. The output of the format mask is converted to Upper case.                      |
| z                | String must contain an alphabetic letter (A-Z, a-z) or a digit (0-9) in this position.                                                                                |
| (nn)             | A numeric value surrounded by parenthesis may be used to specify a repeat count for the preceding format character. For example AAAAAA may also be specified as A(5). |
| <i>other</i>     | Outputs the character.                                                                                                                                                |

# Labels/Logical Statement References

ProvideX supports the use of logical statement references (line labels) in lieu of actual line number/label references in your applications. ProvideX supplies a set of *built-in labels* (keywords, with *leading asterisks*) that can be used wherever you would use a statement reference. Note that some of the logical line labels and the directives they emulate will remove an item from your stack and perform a **RESET**.

The following logical statement references are supported in ProvideX:

|           |                                                                                               |
|-----------|-----------------------------------------------------------------------------------------------|
| *BREAK    | Emulates <b>BREAK Immediate Exit of Loop</b> , <i>p.33</i> .                                  |
| *CONTINUE | Emulates <b>CONTINUE Initiates Next Iteration of Loop</b> , <i>p.57</i> .                     |
| *END      | Emulates <b>END Halt Program Execution</b> , <i>p.113</i> .                                   |
| *ESCAPE   | Emulates <b>ESCAPE Interrupt Program Execution</b> , <i>p.122</i> .                           |
| *NEXT     | Goes to the beginning of the next line/statement.                                             |
| *PROCEED  | Continues to the next statement in a compound statement or to the beginning of the next line. |
| *RETRY    | Emulates <b>RETRY Re-Execute Failing Instruction</b> , <i>p.290</i> .                         |
| *RETURN   | Emulates <b>RETURN Subroutine/Function Return</b> , <i>p.291</i> .                            |
| *SAME     | Goes to the start of current line/statement.                                                  |



**Note:** In earlier versions of ProvideX, the \*CONTINUE and \*BREAK labels and the corresponding directives were not supported for use with **SELECT / NEXT RECORD** directives. **BREAK** and \***BREAK** commands can now be used in **SELECT** structures.

## Examples

```
A=0, Y$=""
READ (1,IND=A++,ERR=*NEXT)X$; Y$+=X$; GOTO *SAME
```

## See Also

**BREAK Directive**, *p.33*  
**CONTINUE Directive**, *p.57*  
**END Directive**, *p.113*  
**ESCAPE Directive**, *p.122*  
**RETRY Directive**, *p.290*  
**RETURN Directive**, *p.291*



## Negative CTL Definitions

ProvideX normally handles all negative CTL values internally. Values are used as follows:

- 1 to -999            used by the input handler to save current instructions, internally call **"\*CONTROL"**
- 1000 to -1999    for input editing control keys and mouse interaction
- 2000 to -2255    for composite character generation


### Table of Negative CTL Values

The negative CTL value and their assigned values are listed as follows:

| <i>CTL Value</i> | <i>Assigned Actions</i>                                                                                                                                            |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -1               | Invoke utility sub-menu                                                                                                                                            |
| -2               | Invoke screen print utility                                                                                                                                        |
| -3               | Reserved                                                                                                                                                           |
| -4               | Display session statistics                                                                                                                                         |
| -5               | Field help                                                                                                                                                         |
| -6               | Field query                                                                                                                                                        |
| -7               | Program help                                                                                                                                                       |
| -8               | Reserved                                                                                                                                                           |
| -9               | Reserved                                                                                                                                                           |
| -10 to -999      | Save current instruction and screen, then call user program \$CTL- <i>nnn</i> ( <i>-nnn</i> = CTL value). Upon exit, reset screen and re-execute saved instruction |
| -1000            | Ignore key or ignore <b>SCROLL WHEEL</b> action                                                                                                                    |
| -1001            | Generate Escape/Break                                                                                                                                              |
| -1002            | Clear input buffer and blank on screen                                                                                                                             |
| -1003            | Backspace and delete prior character                                                                                                                               |
| -1004            | Backup one position to left                                                                                                                                        |
| -1005            | Forward one position to right                                                                                                                                      |
| -1006            | Insert a blank at current position                                                                                                                                 |
| -1007            | Delete character at current position                                                                                                                               |
| -1008            | Skip to next blank character                                                                                                                                       |
| -1009            | Toggle Insert mode                                                                                                                                                 |
| -1010            | Return to start of input (Home)                                                                                                                                    |
| -1011            | Up a line or <b>SCROLL WHEEL</b> up                                                                                                                                |
| -1012            | Down a line or <b>SCROLL WHEEL</b> up                                                                                                                              |
| -1013            | Page up or <b>Ctrl</b> - <b>SCROLL WHEEL</b> up                                                                                                                    |
| -1014            | Page down or <b>Ctrl</b> - <b>SCROLL WHEEL</b> down                                                                                                                |

| <b>CTL Value</b> | <b>Assigned Actions</b>                   |
|------------------|-------------------------------------------|
| -1015            | Tab forward 10 spaces                     |
| -1016            | Tab backward 10 spaces                    |
| -1017            | Return to start end re-input              |
| -1018            | Go to end of input                        |
| -1019            | Shift screen to the left                  |
| -1020            | Shift screen to the right                 |
| -1021            | Advance to start of next word             |
| -1022            | Go back to start of previous word         |
| -1023            | Clear from cursor to end of input         |
| -1024            | Restore input line to original value      |
| -1025            | Go Home and reset default                 |
| -1026 to -1079   | <i>Reserved for future use</i>            |
| -1080            | LEFT-MOUSE-CLICK-DOWN /drag               |
| -1081            | LEFT-MOUSE-CLICK-UP                       |
| -1082            | Reserved for RIGHT-MOUSE-CLICK-DOWN/ drag |
| -1083            | Reserved for RIGHT-MOUSE-CLICK-UP         |
| -1084 to -1098   | <i>Reserved for future use</i>            |
| -1099            | Reject keystroke and ring bell            |
| -1100            | <i>Reserved for future use</i>            |
| -1101            | Lost Focus                                |
| -1102            | Received Focus                            |
| -1103            | Display Cursor                            |
| -1104            | Focus has changed                         |
| -1105            | Window resized                            |
| -1106 to -1199   | <i>Reserved for future use</i>            |
| -1200            | Context-sensitive help                    |
| -1201 to -1299   | <i>Reserved for future use</i>            |
| -1300            | Open Trace Window (Windows only)          |
| -1301            | Open Command Window (Windows only)        |
| -1302            | Open Command Window (Windows only)        |
| -1303            | Open Command Window (Windows only)        |
| -1304 to -1309   | <i>Reserved for future use</i>            |
| -1310            | End trace                                 |
| -1311            | End command window                        |
| -1312            | End watch window                          |
| -1313            | End breakpoint window                     |
| -1314 to -1399   | <i>Reserved for future use</i>            |

| CTL Value      | Assigned Actions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -1401 to -1403 | <p><b>Message Bar Region LEFT-MOUSE-CLICK Events.</b> ProvideX returns CTL values when the user performs a LEFT-MOUSE-CLICK on a segment in the message bar region. Each of the four possible segments of the message bar region has been assigned a different negative CTL value. The event is reported on the button up only.</p> <p>1st area (segment zero) returns CTL= -1400<br/>           2nd area (segment 1) returns CTL= -1401<br/>           3rd area (segment 2) returns CTL= -1402<br/>           4th area (segment 3) returns CTL= -1403</p> <p>See also: <b>'MESSAGE' Mnemonic, p.620.</b></p>   |
| -1404 to -1408 | <i>Reserved for future use</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| -1409          | <p><b>ToolBar LEFT-MOUSE-CLICK-UP</b></p> <p>MOUSE-CLICK CTL events are now supported in the ToolBar bar region. The event is reported on the button up only.</p> <p>RIGHT-MOUSE-CLICK-UP returns CTL= -1409</p>                                                                                                                                                                                                                                                                                                                                                                                                |
| -1410 to -1413 | <p><b>Message Bar Region RIGHT-MOUSE-CLICK Events.</b> ProvideX returns CTL values when the user performs a RIGHT-MOUSE-CLICK on a segment in the message bar region. Each of the four possible segments of the message bar region has been assigned a different negative CTL value. The event is reported on the button up only.</p> <p>1st area (segment zero) returns CTL= -1410<br/>           2nd area (segment 1) returns CTL= -1411<br/>           3rd area (segment 2) returns CTL= -1412<br/>           4th area (segment 3) returns CTL= -1413</p> <p>See also: <b>'MESSAGE' Mnemonic, p.620.</b></p> |
| -1414 to -1418 | <i>Reserved for future use</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| -1419          | <p><b>ToolBar MOUSE-CLICK Events</b></p> <p>MOUSE-CLICK CTL events are now supported in the ToolBar bar region. The event is reported on the button up only.</p> <p>LEFT-MOUSE-CLICK-UP returns CTL= -1409<br/>           RIGHT-MOUSE-CLICK-UP returns CTL= -1419</p>                                                                                                                                                                                                                                                                                                                                           |
| -1420 to -1801 | <i>Reserved for future use</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| -1802 to -1803 | <p><b>RIGHT-MOUSE-CLICK Change.</b> RIGHT-MOUSE-CLICK no longer returns CTL=4.</p> <p>RIGHT-MOUSE-CLICK-DOWN returns CTL= -1802<br/>           RIGHT-MOUSE-CLICK-UP returns CTL= -1803</p> <p>Changed in *DEV/WINDOWS and *DEV/WINTERM</p>                                                                                                                                                                                                                                                                                                                                                                      |
| -1804 to -1899 | <i>Reserved for future use</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| -1900          | Input Time-out (NOMADS Internal)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

| CTL Value      | Assigned Actions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -1999          | <p>User's window_close request. By default, ProvideX maps this to CTL=4 (in *DEV/WINDOWS). You can test for CTL=4 in your programs, or you can use the <a href="#">DEFCTL Directive, p.78</a>, to remap -1999 to a different CTL; e.g.,</p> <pre>DEFCTL -1999=10 ! Maps the &lt;F10&gt; key (CTL=10) for close</pre> <p>You can test the <b>EOM</b> system variable to find out whether the user pressed the actual function key or selected one of the close options (i.e., the Windows  close button). HTA (EOM) returns \$00800004\$ for &lt;Alt-F4&gt; or \$0080F831\$ for the close option. For more information, refer to the <a href="#">'F4' System Parameter, p.665</a>, and the <a href="#">EOM System Variable, p.559</a>.</p> |
| -2000 to -2255 | <p><b>Composite Character Generation.</b> Whenever ProvideX detects a CTL value in this range, the character whose ASCII value equals the absolute value of CTL less 2000 is placed into the input buffer. This feature allows you to define input sequences to generate the Extended ASCII characters even when the terminal cannot generate them.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| -2256 & below  | <p><i>Reserved for future use</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |



**Note:** CTL values less than -32001 or greater than 32001 are reserved for internal use. Please *do not* use CTL values in these ranges.

# Operators

This section describes the traditional operators (e.g., + - \* /) you can use in ProvideX, along with other operators for which support has been added in recent versions (e.g., the auto-increment and decrement operators, assignment operators, the **LIKE Operator**, and the **Apostrophe Operator**). Use of these operators is further described in the *ProvideX User's Guide*.

## Arithmetic Operators

The following operators are used to perform calculations.

- + *Addition*. Where  $A + B$  **adds** A to B.
- *Subtraction*. Where  $A - B$  **subtracts** B from A.
- \* *Multiplication*. Where  $A * B$  **multiplies** A by B.
- / *Division*. Where  $A / B$  **divides** A by B.
- ^ *Raise to*. Where  $A ^ B$  raises A to the **power** of B (\*\* is equivalent to ^).
- | *Modulus*. Where  $A | B$  divides A's **remainder** by B.

In addition to the arithmetic operators, there are *increment* and *decrement* features:

- ++ *Auto-increment* by 1 (pre-increment when prefixed to variable name, post-increment if suffixed); e.g., ++var1 **or** var1++.
- *Auto-decrement* by 1 (pre-decrement when prefixed to variable name, post-decrement if suffixed); e.g., --var1 **or** var1--.

## Relational and Logical Operators

The following operators are used to compare two *numeric or string* values.

- = Where  $A = B$  yields 1 if A and B are equal, else yields 0 zero.
- < Where  $A < B$  yields 1 if A is less than B, else yields 0 zero.
- > Where  $A > B$  yields 1 if A is greater than B, else yields 0 zero.
- <> Where  $A <> B$  yields 1 if the A and B are not equal, else yields 0 zero.
- <= Where  $A <= B$  yields 1 if A is less than or equal to B, else yields 0 zero.
- >= Where  $A >= B$  yields 1 A is greater than or equal to B, else yields 0 zero.
- AND** Where  $A \text{ AND } B$  yields 1 if both values are non-zero, else yields 0 zero.
- OR** Where  $A \text{ OR } B$  yields 1 if either values are non-zero, else yields 0 zero.



*Note:* <>, <=, and >= can be entered as ><, =<, and => respectively.

## Assignment Operators

The following assignment operators are included in the general syntax of the language:

- +=** *Add to.* Can be used with numerics or strings:  
     Numeric example:  $A+=1$  is the same as  $A=A+1$ .  
     String example:  $A\$\ += "G"$  is the same as  $A\$=A\$+"G"$ .
- =** *Subtract from.* Not valid for strings.  
     Numeric only:  $B-=A+1$  is the same as  $B=B-(A+1)$ .
- \*=** *Multiply by.* Can be used with numerics or strings.  
     Numeric example:  $B*=A+1$  is the same as  $B=B*(A+1)$ .  
     String example:  $A\$*=5$  is the same as  $A\$=A\$+A\$+A\$+A\$+A\$$ .
- /=** *Divide by.* Not valid with strings.  
     Numeric only:  $B/=A+1$  is the same as  $B=B/(A+1)$ .
- ^=** *Exponentiation.* Raise to. Not valid with strings.  
     Numeric only:  $B^=A$  is the same as  $B=B^A$ .
- |=** *Modulus / remainder from division.* Not valid with strings.  
     Numeric only:  $B|=A$  is the same as  $B=MOD(B,A)$ .

## LIKE Operator

Use the **LIKE** operator for string comparisons. The ProvideX default is to take the string expression on the *left* and compare it to the string mask on the *right*. You can apply all the regular expression rules of the **MSK( ) Function**, [p.486](#); e.g.,

**IF A\$ LIKE mask THEN ..**

*is the same as ...*

**IF MSK(A\$, mask) <> 0 THEN ..**

This operator may be used in conditional structures, such as **IF.THEN..ELSE**, **REPEAT..UNTIL**, and **WHILE..WEND**.

To make conversions from *Thoroughbred* easier, set the 'TL' parameter to **ON**. With the 'TL' parameter set, the **LIKE** operator emulates the *Thoroughbred* matching of patterns. For more information, refer to the **'TL' System Parameter**, [p.689](#).

## Apostrophe Operator

The apostrophe operator is used to assign, retrieve, list, and make dynamic changes to a given control or object's properties. For full details on the syntax and use of the apostrophe, refer to the **Apostrophe Operator** in the next section.

# Apostrophe Operator

## Formats

1. *Assign Property*: `obj_id'property[$]=var[$]`
2. *Retrieve Property*: `var[$]=obj_id'property[$]`
3. *Call a Method*: `var[$]=obj_id'method[$](args)`
4. *List of Available Properties or Methods*: `var$=obj_id'*`

## Where

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| '                   | <i>Apostrophe Operator</i> (sometimes called a <i>tick</i> )                                                                                                                                                                                                                                                                                                                                                                                                             |
| *                   | Asterisk to produce a comma-separated list of properties/methods available for a particular object; e.g., to list <b>DROP_BOX</b> properties (assuming <code>dbox</code> is a unique <code>ctl_id</code> ):<br><br><code>PRINT dbox' *</code><br><code>Auto,BackColour,Col,Cols,CurrentItem,CtlName,Enabled, Eom,Focus,Font,Height,hWnd,Item,ItemCount,ItemText,Key,Left,Line,Lines,Msg,OnFocusCtl,Parent,Sep,SepLoad,Tbl,TextColour,Tip,Top,Value,Visible,Width,</code> |
| <i>args</i>         | Optional argument(s).                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>method[\$]</i>   | Name of a valid method/function in the given control/object. Method names are not case-sensitive. To query the list of available methods, use the syntax for asterisk ' * described above                                                                                                                                                                                                                                                                                |
| <i>obj_id</i>       | Numeric variable containing the object identifier (handle) for the object.                                                                                                                                                                                                                                                                                                                                                                                               |
| <i>property[\$]</i> | Name of a valid property in the given control/object. Property names are not case-sensitive. To query the list of available properties, use the syntax for asterisk ' * described above.                                                                                                                                                                                                                                                                                 |
| <i>var[\$]</i>      | String or numeric variable.                                                                                                                                                                                                                                                                                                                                                                                                                                              |



**Note:** In the above syntax, ensure that numeric methods/properties correspond to numeric variables and that string methods/properties correspond to string variables; i.e., `obj_id'property$=var$` or `var=obj_id'method()` on both sides of the equation.

## Description

An apostrophe operator (*tick*) allows *dynamic* access to the properties and methods available for a given COM, OOP, or graphical object. While the syntax is generally the same for all object-oriented coding in ProvideX, the apostrophe operator can be used to read and alter properties, or execute methods, in a variety of different control and object types.

For detailed information on the use of the apostrophe operator for specific application development purposes, refer to the following sources:

- ProvideX *Event Handling* documentation, *Automation in ProvideX*.
- ProvideX *NOMADS Reference* and related *Direxions* presentations.
- [Chapter 7. Control Object Properties, p.701.](#)
- [Object Oriented Programming, p.22.](#)

## Examples

The first example creates a *multi-line control*, displays the properties available, changes the column width, and returns the current screen coordinates.

```
->multi_line 100,@(10,12,40,1)
->LET X=100
->PRINT X' *
Auto,BackColour,Col,Cols,CtlName,Enabled,Eom,Fmt,Focus,Font,Height,hWnd,
ImpliedDecimal,Key,Left,Len,Line,Lines,Lock,MenuCtl,Msg,Nul,OnFocusCtl,
Parent,Scroll,SelectLength,SelectOffset,SelectText,Sep,SignalOnExit,
TextColour,Tip,Top,Uppercase,Value,Visible,Width,
->LET X'Cols=50 ! Make control 50 columns wide
->PRINT X'Col,X'Line ! Current screen coordinates of the multi-line
 10 12
->
```

The following example creates a *grid* containing 10 columns and 5 rows, then selects column two, row zero (which selects the entire column). Changing the value of the property sets the contents of all the selected cells to that value.

```
->GRID 10,@(10,10,40,5)
->Y=10
->Y'ColumnsWide=10
->Y'RowsHigh=5
->Y'Column=2
->Y'Row=0
->Y'Value$="New Data"
```

This dynamically changes a property in one object based on the value of another:

```
IF Country ctl'value$ = "CDN" \
 THEN Zip'Fmt$="A0A 0A0" \
 ELSE Zip'Fmt$="00000"
```

The following is a simple *COM interface* example that instantiates an Internet Explorer object, then displays the PVX website:

```
DEF OBJECT IE, @(10,10,40,20)="Shell.Explorer"
IE'Navigate2('www.pvx.com')
```

The following OOP example assumes the definition of a "Customer" object:

```
Cst= NEW ("Customer")
Cst'Find("012345")
```



# System Limits

This section lists ProvideX system limits (subject to operating system constraints on memory and resources). There is also a general limit of **2GB** on all data storage.

The following are preset limits for all ProvideX *programs*:

| <b>For ProvideX Basic Programs</b>                       | <b>Limit</b> | <b>Notes</b>                                             |
|----------------------------------------------------------|--------------|----------------------------------------------------------|
| Array (32-bit platforms), <i>max elements (millions)</i> | 10           |                                                          |
| Array, <i>dimensions</i>                                 | 3            |                                                          |
| Command line (32-bit platforms), <i>max characters</i>   | 32767        | Entire length includes path + all parameters, arguments. |
| Command line (32-bit platforms), <i>max arguments</i>    | 128          | Includes path, program name + all arguments.             |
| COM interface method calls, <i>max arguments</i>         | 20           |                                                          |
| Directory name, <i>max characters</i>                    | 260          |                                                          |
| File name, <i>max characters</i>                         | 511          | Includes path + file name.                               |
| Line number, <i>highest</i>                              | 64999        |                                                          |
| Line label, <i>max characters</i>                        | 127          |                                                          |
| Memory, <i>max GB</i>                                    | 2            | ... or the maximum the OS allows, whichever smaller.     |
| Mnemonics, <i>max characters (billion)</i>               | 2            | String length for data                                   |
| Precision, <i>digits of accuracy</i>                     | 18           | The default is 2.                                        |
| Program name (arguments), <i>max characters</i>          | 2044         | Includes path + file name + line label + data.           |
| Program size (32-bit platforms), <i>max GB</i>           | 2            | ... or the maximum the OS allows, whichever smaller.     |
| Record limit, <i>default max bytes</i>                   | 10240        |                                                          |
| Statement length, <i>max in KB</i>                       | 24           |                                                          |
| Statement length (console editing), <i>max in KB</i>     | 4            |                                                          |
| String size, <i>max GB</i>                               | 2            | Refer to <b>Internal Limitations</b> below.              |
| TCP sockets (UNIX), <i>max connections</i>               | 65535        |                                                          |
| TCP sockets (Windows Server), <i>max connections</i>     | 65535        |                                                          |
| TCP/IP address, <i>max characters</i>                    | 60           |                                                          |
| Variable name, <i>max characters</i>                     | 127          |                                                          |
| WebServer port, <i>highest port number</i>               | 65535        | Values < 2000 reserved for standard Internet activities  |
| Window <i>character limit</i>                            | 255          | Max. 225 for lines.<br>Max. 255 for columns.             |

The following are preset limits for *data files*:

| <i>For Data Files</i>                           | <i>Limit</i> | <i>Notes</i>                       |
|-------------------------------------------------|--------------|------------------------------------|
| Channels, <i>max number</i>                     | 127/65000    | See 'XF' System Parameter, p.695   |
| File size, <i>max bytes</i>                     | $2^{31-1}$   | See 'MB'= System Parameter, p.674  |
| Files, <i>max number open concurrently</i>      | 500          |                                    |
| Key segments, <i>max KB</i>                     | 48 - 96      |                                    |
| Key I/O buffers, <i>max number</i>              | 100          |                                    |
| Keys, <i>max number of</i>                      | 16/256       | 0-15 for VLR/FLR, or 0-255 for EFF |
| Keys, secondary alternate, <i>max length</i>    | 240          |                                    |
| Record size (fixed), <i>max bytes</i>           | 32167        |                                    |
| Record size (variable length), <i>max bytes</i> | 31000        |                                    |

### *Internal Limitations*

With this increased string limit a few internal logical limitations have been imposed to avoid passing excessively long strings to some of the system functions. The following functions/directives are restricted to 8kb string lengths:

- **CALL** subprogram names/entry point names.
- **LIST\_BOX LOAD** for a single line.
- **SYSTEM\_HELP** command.
- **STR( )** function format masks.
- **SYS( )** function system command.
- **XEQ( )** pathname.
- **EVS( )/EVN( )** expressions.
- **MSK( )** mask definition.
- **DAY\_FORMAT** and **DTE( )** formats.

# Reserved Words

This words on this list are reserved for internal use by ProvideX. *Do not use them as variable names.*

|            |               |             |             |               |               |
|------------|---------------|-------------|-------------|---------------|---------------|
| ABS        | DISABLE       | HIDE        | MID         | QUO           | SSZ           |
| ACCEPT     | DLL           | HLP         | MIN         | RADIO_BUTTON  | START         |
| ACS        | DLM           | HSA         | MNEMONIC    | RANDOM        | STEP          |
| ADD        | DLX           | HSB         | MNM         | RANDOMIZE     | STK           |
| ADDR       | DOM           | HTA         | MNU         | RCD           | STOP          |
| ALL        | DROP          | HWD         | MOD         | RDX           | STP           |
| AND        | DROP_BOX      | HWN         | MSE         | READ          | STR           |
| ARG        | DSK           | H_SCROLLBAR | MSG         | REC           | SUB           |
| ASC        | DSZ           | I3E         | MSGBOX      | RECORD        | SWAP          |
| ASN        | DTE           | I86         | MSK         | REFILE        | SWITCH        |
| ATH        | DUMP          | IF          | MSL         | RELEASE       | SWP           |
| ATN        | EDIT          | IND         | MULTI_LINE  | REM           | SYS           |
| AUTO       | ELSE          | INDEX       | MULTI_MEDIA | REMOVE        | SYSTEM_HELP   |
| AUTO_LOCK  | ENABLE        | INDEXED     | MXC         | RENAME        | SYSTEM_JRNL   |
| BEGIN      | END           | INPUT       | MXL         | RENUMBER      | TABLE         |
| BIN        | ENDTRACE      | INT         | NAR         | REPEAT        | TAN           |
| BKG        | END_IF        | INVOKE      | NBF         | RESET         | TBL           |
| BREAK      | ENTER         | IOL         | NEXT        | RESTORE       | TCB           |
| BSY        | ENV           | IOLIST      | NID         | RET           | THEN          |
| BSZ        | EOM           | IOR         | NOT         | RETRY         | TIM           |
| BUTTON     | EPT           | ISZ         | NUL         | RETURN        | TIP           |
| BYE        | ERASE         | JUL         | NUM         | RND           | TME           |
| CALL       | ERR           | KEC         | OBJ         | RNO           | TMR           |
| CASE       | ERROR_HANDLER | KEF         | OBJECT      | ROUND         | TMS           |
| CHART      | ERS           | KEL         | OBTAIN      | RSZ           | TO            |
| CHECK_BOX  | ESC           | KEN         | OFF         | RTY           | TRANSLATE     |
| CHG        | ESCAPE        | KEP         | ON          | RUN           | TRISTATE_BOX  |
| CHN        | EVN           | KEY         | OPEN        | SAME          | TRX           |
| CHR        | EVS           | KEYED       | OPT         | SAVE          | TSK           |
| CLEAR      | EXCEPT        | KGN         | OR          | SCALL         | TSM           |
| CLIP_BOARD | EXECUTE       | KNO         | OWN         | SECURITY_MASK | TXH           |
| CLOSE      | EXIT          | LAOD        | PAD         | SELECT        | TXW           |
| CONTINUE   | EXP           | LCS         | PASSWORD    | SEP           | UCS           |
| CONTROL    | EXTRACT       | LEN         | PCK         | SERIAL        | UID           |
| COS        | FFN           | LET         | PERFORM     | SERVER        | UNLOCK        |
| CPL        | FI            | LFA         | PFX         | SETCTL        | UNT           |
| CRC        | FIB           | LFO         | PGM         | SETDAY        | UNTIL         |
| CREATE     | FID           | LIKE        | PGN         | SETDEV        | UPK           |
| CSE        | FILE          | LINE_SWITCH | POINT       | SETDRIVE      | USER_LEX      |
| CTL        | FIN           | LIP         | POP         | SETERR        | VAL           |
| CUSTOM_VBX | FIND          | LIST        | POPUP_MENU  | SETESC        | VARDROP_BOX   |
| CVS        | FLG           | LIST_BOX    | POS         | SETFID        | VARLIST_BOX   |
| CWDIR      | FLOATING      | LOAD        | PRC         | SETMOUSE      | VIA           |
| DATA       | FMT           | LOCAL       | PRECISION   | SETTIME       | VIDEO_PALETTE |
| DAY        | FN            | LOCK        | PREFIX      | SETTRACE      | VIN           |
| DAY_FORMAT | FNT           | LOG         | PREINPUT    | SET_FOCUS     | VIS           |
| DEC        | FOR           | LONG_FORM   | PRINT       | SET_NBF       | V_SCROLLBAR   |
| DEF        | FPT           | LPG         | PRM         | SET_PARAM     | WAIT          |
| DEFAULT    | FROM          | LRC         | PROCEED     | SGN           | WEND          |
| DEFCTL     | GAP           | LSIT        | PROCESS     | SHORT_FORM    | WHERE         |
| DEFPRT     | GBL           | LST         | PROGRAM     | SHOW          | WHILE         |
| DEFTTY     | GEP           | LWD         | PROPERTIES  | SID           | WHO           |
| DELETE     | GET_FILE_BOX  | MAX         | PROPERTY    | SIN           | WINDOW        |
| DICTIONARY | GFN           | MDE         | PSZ         | SIZ           | WINPRT_SETUP  |
| DIM        | GID           | MEM         | PTH         | SORT          | WRITE         |
| DIR        | GO            | MENU_BAR    | PUB         | SQR           | XEQ           |
| DIRECT     | GRID          | MERGE       | PURGE       | SRT           | XFA           |
| DIRECTORY  | HFN           | MESSAGE_LIB | QUIT        | SSN           | XOR           |

# Error Codes and Messages

- 0: Record/file busy
- 1: Logical END-OF-RECORD reached
- 2: END-OF-FILE on read or File full on write
- 3: Input/Output error on file
- 4: Device not ready
- 5: Data error on device or file
- 6: Directory error
- 7: Access out of file boundaries
- 8: Data write error
- 9: Unable to restore calling program
- 10: Illegal pathname specified
- 11: Record not found or Duplicate key on write
- 12: File does not exist (or already exists)
- 13: File access mode invalid
- 14: Invalid I/O request for file state
- 15: Operating system command failed
- 16: File/Disc is full
- 17: Invalid file type or contents
- 18: Program not loaded/Invalid program format
- 19: Program size too large
- 20: Syntax error
- 21: Statement number is invalid
- 22: Invalid compound statement
- 23: Missing/Invalid variable
- 24: Attempt to duplicate a function name
- 25: Invalid call to user function (Non-existent or recursive)
- 26: Variable type invalid
- 27: Unexpected or incorrect WEND, RETURN, or NEXT
- 28: No corresponding FOR for NEXT
- 29: Invalid Mnemonic or position specification
- 30: Statement too complex -- cannot compile
- 31: Memory limits reached -- Increase 'SZ' option
- 32: Invalid or redundant Input/Output option
- 33: Insufficient memory available in system -- try later
- 34: Directive not allowed from COMMAND mode
- 35: Invalid date/time specified
- 36: ENTER parameters don't match those of the CALL
- 37: Directive can only execute in subprogram
- 38: Directive cannot be used within CALLED program
- 39: Invalid record definition
- 40: Divide check or numeric overflow
- 41: Invalid integer encountered (range error or non-integer)
- 42: Subscript out of range/Invalid subscript
- 43: Format mask invalid
- 44: Invalid step value
- 45: Referenced statement invalid
- 46: Length of string invalid
- 47: Substring reference out of string
- 48: Invalid input -- Try again
- 49: <\*> Internal program format error <\*>
- 50: Reserved for FUTURE USE
- 51: Invalid VFU/key load
- 52: Program is password protected
- 53: Invalid password
- 54: Unable to load ERROR HANDLER
- 55: Cannot locate statement label
- 56: Duplicate statement label
- 57: No such window defined
- 58: Line(s) in GOSUB/FOR/WHILE stack
- 59: Invalid directive in function/object definition
- 60: Invalid control argument value
- 61: Authorization failure
- 62: Not a development system
- 63: Not activated for this software package
- 64: No valid LEX table loaded
- 65: Window element does not exist or already exists
- 66: Warning-Program size > 64K -- may not run on all environments
- 67: VBX processor reported a failure
- 68: RPC (Remote Process Call) name not found
- 69: No Journalization file open
- 70: EDIT command syntax error
- 71: String not found
- 72: Replacement string will not fit
- 73: No current string defined
- 74: RENUMBER rejected -- Line numbers too large
- 75: Invalid Hex string
- 76: LINE\_SWITCH failure - Terminal cannot be switched
- 77: Edit generates no line number
- 78: Invalid MSK specification
- 79: Invalid FORMAT specification
- 80: Invalid key definition, number or name
- 81: Invalid IOLIST specification
- 82: File must be 'LOCKED' before being 'PURGED'
- 83: Invalid statement number range
- 84: No DICTIONARY exists on OPENed file
- 85: Program does not support line numbers..
- 86: Transmission error to device
- 87: MENUBAR definition invalid
- 88: Invalid/unknown property name
- 89: File access denied -- I/O operation pending
- 90: Unable to locate Object class definition
- 91: Class/Object in use
- 92: Invalid CLASS definition
- 93: Already defined within class definition
- 94: Loop in Class inheritance found
- 95: Bad Object Identifier
- 96: Invalid Return Value
- 97: Version conflict - function not supported
- 98: Feature not yet implemented
- 99: Feature not supported
- 100: No driver for terminal type or library missing
- 101 - 102: No message
- 105 - 114: Keyed file errors
- 115: File I/O Verification Error
- 116: Invalid field descriptor byte
- 117: Invalid segment number
- 118: Keyed file error
- 119: No message.
- 120: Internal system logic error
- 121: Invalid program format
- 122: No message.
- 123: Warning-The following statement labels cannot be located
- 124: Warning-The following statement labels occur more than once
- 125: Improper Structure Detected
- 126: Forced termination - No valid activation file
- 127: Break key depressed
- >256: Operating System Errors



**Note:** The hyperlist above is linked to the complete list of error codes. The error and warning messages are explained in further detail on the *following pages*.

# List of Messages

Whenever ProvideX encounters an error, it sets the **ERR System Variable**, *p.560*, to an error code (an integer). The associated error message indicates the type of error.

**ERR** values greater than 255 indicate operating system errors. In these cases, ProvideX returns the value of the operating system's error code (integer) plus 256. Use the **MSG( ) Function**, *p.484*, to obtain the error message that is associated with the number:

```
->?msg(4)
Error #4: Device not ready
```

This section provides a numerically arranged list of all the current error codes and their meanings.

Error #0: Record/file busy

*Possible Reasons:*

- Cannot open a file that is locked by another user.
- Cannot **READ**, **FIND**, **EXTRACT**, or **INPUT** if a record is being extracted by another user.
- Time-out occurred on a device.
- Permission denied.

The `BSY=stmtref` option allows you to trap Error #0: Record/file busy.

Error #1: Logical END-OF-RECORD reached

Combined length of data elements cannot exceed preset maximum record length (as defined for your given file).

Error #2: END-OF-FILE on read or File full on write

*Possible Reasons:*

- On a read: the end of the file has been reached or
- On a write: the file is full or has reached a preset maximum record count.
- [TCP] disconnection

When ProvideX is processing a **CLOSE** directive and Error #2 is reported because of a full disk, the error is only reported once. Then ProvideX internally trashes the pending data and closes the channel.

Error #3: Input/Output error on file

A physical (hardware) error was returned from a device. If errors are recurring on a disk drive, record and report them to your hardware maintenance supplier.

Error #4: Device not ready

A "not ready" status was returned from the device. If the device is a printer, see if it is out of paper or the off-line button has been pressed.

**Error #5: Data error on device or file**

Typically reports a hardware malfunction: an error has occurred on a device during a read or write (most often on a read, indicating that the system is unable to read the data correctly).

**Error #6: Directory error**

Unable to re-establish a lock on a file.

**Error #7: Access out of file boundaries**

On a read or write – cannot use a record index which exceeds the maximum number of records allowed for the file.

**Error #8: Data write error**

Typically reports a hardware malfunction: an error has occurred while trying to update a data file.

**Error #9: Unable to restore calling program**

Legacy DOS only – unable to reload a program that was removed from memory (due to memory limitations).

**Error #10: Illegal pathname specified**

Invalid filename on a **SAVE**, **OPEN**, **LOAD**, **RUN**, or **CALL** directive. Possible reasons: filename too long, contains invalid characters, or has invalid syntax. A null string is an invalid pathname.

In **[TCP]** – an invalid IP or DNS address.

**Error #11: Record not found or Duplicate key on write****Possible Reasons:**

- Nonexistent record specified in **KEY=** or **IND=** option. On a **READ** or **EXTRACT** directive (**KEY=** option only) – index position goes to the record with the next higher record key. Use the **DOM=** option to trap and process this error.
- **[TCP]** file I/O includes an **IND=value** or **KEY=value** for a client that is not connected.
- Menu bar item cannot be found when processing sub-commands; i.e., **ON/OFF/ENABLE/DISABLE**.

**Error #12: File does not exist (or already exists)****Possible Reasons:**

- Cannot create a file (**DIRECT**, **INDEXED**, **SORT**, etc.) if a file of the same given name already exists.
- Filename does not exist – cannot **ERASE**, **OPEN**, **LOAD**, **RUN**, or **CALL** a non-existent file.
- In **[TCP]** – can't **OPEN**, server is not listening.

**Error #13: File access mode invalid***Possible Reasons:*

- Cannot send output to serial file unless first locked.
- Cannot receive input from an output-only device (e.g., a printer).
- Cannot send output to an input only device,
- Cannot write given output to file without a **KEY=** option or not before extracting a record.
- Cannot write to a non-externally keyed file using a **KEY=** option.
- Cannot drop the only active window on a terminal.
- OS returned an access mode violation or a permission denied status on an **OPEN INPUT** or **OPEN PURGE** directive for a file.
- Attempt to apply/remove a password when the file is in read-only mode, not locked, or not empty.

**Error #14: Invalid I/O request for file state***Possible Reasons:*

- Cannot access a file that is not opened.
- Cannot unlock a file that is not locked.
- Cannot open a logical file number (channel) that is already open
- On a **CLOSE** directive – cannot close a channel that is not open, but no message is returned unless you used the **ERR=** option.
- Attempt to apply a password to an unopened channel.

**Error #15: Operating system command failed***Possible Reasons:*

- Returned from the operating system. For more details **PRINT MSG(-1)**. One possible reason: exhaustion of .bmp handles.
- In **[TCP]**, can't open the socket locally or general TCP failure on file I/O.

**Error #16: File/Disc is full**

Typically, either the disk is full or the user has reached an allotted limit. The file being written to has no additional room for expansion.

**Error #17: Invalid file type or contents***Possible Reasons:*

- Cannot use a **KEY=** option when the file does not have a key.
- Cannot have unequal lengths for the two parameters for an **AND()**, **IOR()**, or **XOR()** function.
- Cannot **SAVE**, **LIST**, or **MERGE** a file that has a key.
- Cannot **WRITE** to a program file.
- Attempt to apply a password to a non-keyed file.
- Attempt to encrypt a non-VLR formatted file.

Error #18: Program not loaded/Invalid program format  
Cannot **LOAD**, **RUN**, or **CALL** a file which is not in the correct program format.

Error #19: Program size too large  
Program exceeds the 64k limit

Error #20: Syntax error  
The program statement contains a syntax error (usually just a typing error).

Error #21: Statement number is invalid

*Possible Reasons:*

- Cannot use a line number in excess of maximum 65000.
- Cannot edit/append to a non-existent line.
- Cannot refer incorrectly to an existing line  
e.g., **TBL=nnnn** where *nnnn* is not a table.
- Program does not have line numbers.

Error #22: Invalid compound statement  
Directive out of position: the particular directive *must be the final item* in a compound statement because it has the potential to transfer control; e.g., **GOTO**, **GOSUB**, etc.

Error #23: Missing/Invalid variable  
Mandatory variable is missing from the statement's syntax.

Error #24: Attempt to duplicate a function name  
Cannot create function of the same given name as one that already exists.

Error #25: Invalid call to user function (Non-existent or recursive)

*Possible Reasons:*

- Cannot call non-existent user-defined function (not defined or name misspelled)
- Cannot exceed limit on number of recursive calls to a user defined function.  
(Maximum of 10 recursive calls, ten levels deep.)

Error #26: Variable type invalid

*Possible Reasons:*

- Cannot use numeric value in string variable.
- Cannot use string value in numeric variable.

Error #27: Unexpected or incorrect **WEND**, **RETURN**, or **NEXT**  
Cannot execute **NEXT**, **WEND**, **RETURN**, or **UNTIL** if there is no corresponding entry at the top of the stack (applies to **WHILE/WEND**, **GOSUB/RETURN**, **FOR/NEXT**, or **REPEAT/UNTIL** stack).



- Error #28: No corresponding FOR for NEXT  
The variable name for the **NEXT** directive does not match the variable name given for the current **FOR** directive.
- Error #29: Invalid Mnemonic or position specification  
*Possible Reasons:*
- Unrecognized or invalid mnemonic (undefined or misspelled, etc.) on a **PRINT** or **INPUT** statement (only generated if you have already set the '**EG**' mnemonic).
  - Invalid position in **@()** function (outside of current window/scroll region boundaries).
- Error #30: Statement too complex -- cannot compile  
Expression cannot exceed 249 parentheses.
- Error #31: Memory limits reached -- Increase '-SZ' option  
Cannot exceed preset maximum size for user work space (set either using a **-SZ** option or in the **START** directive). Free some of the work space (delete unused variables, arrays, etc.) or increase the memory limits.
- Error #32: Invalid or redundant Input/Output option  
Invalid or duplicate option used on the Input/Output directive.
- Error #33: Insufficient memory available in system -- try later  
The operating system has denied a request for additional memory. Either the amount of memory being requested exceeds system capacity or the memory is in use.
- Error #34: Directive not allowed from COMMAND mode  
The particular directive cannot be executed from **COMMAND** mode. The **FOR** directive is one example.
- Error #35: Invalid date/time specified  
An invalid date or time has been specified on a **SETDAY** or **SETTIME** directive.
- Error #36: ENTER parameters don't match those of the CALL  
The parameters in an **ENTER** directive do not match the parameters in the corresponding **CALL** directive. The number and type (numeric or string) must be the same in both directives.
- Error #37: Directive can only execute in subprogram  
Cannot execute an **ENTER** or **EXIT** directive except in subprogram.
- Error #38: Directive cannot be used within CALLED program  
The particular directive cannot be used in a subprogram.
- Error #39: Invalid record definition  
This message applies to ODBC data.

- Error #40: Divide check or numeric overflow
- Cannot divide by zero. (To return zero for divide by zero errors, set the '**D0**' parameter **ON**.)
  - Cannot exceed the machine's limits on the result of an arithmetic operation.
- Error #41: Invalid integer encountered ( range error or non-integer )
- Either the value specified is not an integer or it exceeds the range allowed for the type of operation being performed; e.g.,
- Cannot use logical file number (channel) outside of permitted range.
  - Cannot set **PRECISION** to value outside of range -1 to +18.
  - Cannot exceed 32767 elements in an array.
  - Cannot use line or column position **@...()** function outside of preset window or screen boundaries.
  - Cannot return function value outside of existing range, for instance, a non-existent **TSK()** number.
- Error #42: Subscript out of range/Invalid subscript
- The value of the subscript is beyond the minimum/maximum settings defined for the array.
- Error #43: Format mask invalid
- The format mask in a **PRINT** statement or a **STR()** function is not large enough to contain the numeric value. Use the '**FI**' parameter to suppress the message.
- Error #44: Invalid step value
- Cannot use zero (0) for the **STEP** value in a **FOR** directive.
- Error #45: Referenced statement invalid
- Incorrect reference to existing statement; e.g., **IOL=nnnn** where *nnnn* does not contain an **IOList** or **TBL=nnnn** where *nnnn* does not contain a **TABLE** directive).
- Error #46: Length of string invalid
- This error is reported when the length of the string provided is not appropriate for the function of directive being executed; e.g,
- Format mask is in excess of 8192 bytes.
  - Length of the string in the **KEY=** option cannot exceed the key length for the given file.
  - Cannot execute the **ASC()** function on a null string.
- Error #47: Substring reference out of string
- Possible reasons:*
- Substring does not exist in string variable, or
  - Substring exceeds the capacity of the variable.

**Error #48: Invalid input -- Try again**

This error occurs when data input is checked against a validation list. It indicates one of the following:

- Value for numeric variable is not in the range specified or has too many decimals.
- Value for a string variable exceeds the maximum length allowed.

**Error #49: <\*> Internal program format error <\*>**

The compiled statement contains invalid data and cannot be processed or decompiled. This is either because of an internal ProvideX error or because the program code has been modified externally.

***Possible Reasons:***

- The program on disk has been damaged.
- The program was created using a feature from a newer version of ProvideX which is not supported by the version you are using.
- A new version of ProvideX was installed without the correct Lex tables (/pvx/lib/\_lex\*.\*)

**Error #50: Reserved for FUTURE USE****Error #51: Invalid VFU/key load**

The VFU is not loaded because the mnemonic sequence is incorrect or other than numeric data has been inserted between the 'SL' and 'EL' printer mnemonics.

**Error #52: Program is password protected**

The current program has been saved with a password.

- Cannot change or list the program without using the **PASSWORD** directive.
- Cannot save a passworded program to a serial file.

**Error #53: Invalid password**

The password entered does not match the password recorded for the program or data file. (Sage Software Canada Ltd. can recover a program if the password is unknown.)

**Error #54: Unable to Load Error Handler**

The system is unable to properly load and execute the specified error handler. This commonly happens when an application attempts to open more files than the O/S permits which triggers an un-trapped error. When ProvideX attempts to access the error handler program from disk to report the error, it is unable to do so as this requires a file handle which is what caused the original un-trapped error.

**Error #55: Cannot locate statement label**

Non-existent or misspelled line label during program execution. (If the label is missing during a **SAVE** operation, the program is saved without a reported error but the program may not function properly.)

**Error #56: Duplicate statement label**

This error is only generated by the **SAVE** command. It warns that a line label has been defined twice. The **SAVE** operation is completed without error, but the program may not function properly.

**Error #57: No such window defined***Possible Reasons:*

- Non-existent window number: cannot locate for a **'DROP'**/**'GOTO'**
- Cannot exceed window number 248 in creating new Window.

**Error #58: Line(s) in GOSUB/FOR/WHILE stack**

One of the program lines you are trying to edit is currently in the stack. You must reset or end the program before making the change.

**Error #59: Invalid directive in function/object definition**

In a multi line function, cannot include a directive which could reset the stack or return to program Command mode.

**Error #60: Invalid control argument value**

An attempt was made to access or set an invalid property, or apply an invalid value to a property normally associated with a **GRID** or **CHART**.

**Error #61: Authorization failure***Possible Reasons:*

- Cannot run an unauthorized program or system function. Contact your dealer to obtain the proper activation keys.
- Password record may have failed the internal CRC check.

**Error #62: Not a development system**

Cannot create (or change access to) a secured program. Only developers can perform this task.

**Error #63: Not activated for this software package**

Cannot save a program using a package ID without an activation key.

**Error #64: No valid LEX table loaded**

The system is unable to locate or read the file containing the LEX (syntax) table (\*lextbl.en).

**Error #65: Window element does not exist or already exists**

Cannot reference non-existent window or GUI object that has not yet been drawn.

**Warning #66: Program size > 64K -- may not run on all environments**

ProvideX issues this warning when you save a program that is getting close to the 64k limit.

- Error #67: VBX processor reported a failure  
The VBX Processor reported back with an error.
- Error #68: RPC (Remote Process Call) name not found  
Cannot issue a call to a remote system that has not yet been defined.
- Error #69: No Journalization file open  
File marked to be journalized but no journal file is open.
- Error #70: EDIT command syntax error  
The **EDIT** directive was incorrectly entered. No changes will be made to the program.
- Error #71: String not found  
The string given (enclosed in square brackets) has not been found.
- Error #72: Replacement string will not fit  
The length of the string following an **EDIT 'R'** directive exceeds the number of characters remaining in the original line. The complete **EDIT** command is rejected.
- Error #73: No current string defined  
Cannot reference the current string using [ ] because no current string has been defined.
- Error #74: RENUMBER rejected -- Line numbers too large  
Cannot renumber if the result will generate statements exceeding maximum 65000.  
Change the **RENUMBER** directive to reduce the highest line number used.
- Error #75: Invalid Hex string  
Cannot use characters other than 0 to 9 or A to F in a Hex string.
- Error #76: LINE\_SWITCH failure - Terminal cannot be switched  
The system cannot properly switch file 0 zero to the specified file (cannot switch to a file that is not a terminal).
- Error #77: Edit generates no line number  
On **EDIT** directive - resultant string does not contain a valid leading line number.
- Error #78: Invalid MSK specification  
*Possible Reasons:*
- The mask specified in the **MSK( )** function is invalid.
  - Cannot reuse as current mask if no previous mask already available.
- Error #79: Invalid FORMAT specification  
The IOList contains an invalid format specification.
- Error #80: Invalid key definition, number or name  
*Possible Reasons:*
- Invalid key definition on a **KEYED** directive

- Invalid key number referenced

Error #81: Invalid IOLIST specification

The IOList contains an invalid specification.

Error #82: File must be 'LOCKED' before being 'PURGED'

File must be locked before using a **PURGE** directive to erase all data.

Error #83: Invalid statement number range

Cannot have invalid range of statement numbers; e.g., an ending statement number less than the starting statement number.

Error #84: No DICTIONARY exists on OPENed file

No embedded data dictionary exists but the file **INPUT** directive is looking for one.

Error #85: Program does not support line numbers..

Returned if trying to use line numbers when system parameter '**NN**' set to prohibit line numbers

Error #86: Transmission error to device

A communications problem is reported between a host and a WindX client (usually when the client does not respond quickly).

Error #87: MENUBAR definition invalid

Syntax error in a menubar definition

Error #88: Invalid/unknown property name

The property or method name of a ProvideX object (OOP object/OCX/COM/VBX control or GUI control) is invalid, or the parameters passed to a method call are incorrect.

Error #89: File access denied -- I/O operation pending

Processing an OCX event during the middle of a file I/O operation; e.g., if a program is reading from a TCP/IP port and an OCX event occurs, the event processing logic may not access the TCP/IP port since doing so may harm a pending I/O operation.

Error #90: Unable to locate Object class definition

Attempting to load an object class definition the system did not detect the **DEF CLASS** directive.

Error #91: Class/Object in use

Attempting to **DROP** a class definition while it is still in use.

Error #92: Invalid CLASS definition

Incorrect object class definition. It is likely that an invalid directive was found between the **DEF CLASS** and **END DEF**.

- Error #93: Already defined within class definition  
Two or more **PRECISION** or **PROGRAM** declarations in one class definition.
- Error #94: Loop in Class inheritance found  
Object class being defined is attempting to inherit a class definition that inherits the object class being defined.
- Error #95: Bad Object Identifier  
Specified object handle is not valid or the object has been deleted.
- Error #96: Invalid Return Value  
Invalid return value.
- Error #97: Version conflict - function not supported  
Attempting to use a feature of the language that has been disabled due to use of an activation key for an earlier version of the software. The key you are using only provides access to the functionality that existed at the time the key was issued. You must purchase an newer version of the software.
- Error #98: Feature not yet implemented  
Cannot use the particular function or directive because it isn't implemented in this release of ProvideX.
- Error #99: Feature not supported  
Cannot use the particular function or directive because it's not implemented or available on this hardware platform.
- Error #100: No driver for terminal type or library missing  
During initialization -ProvideX could not locate the device driver module for the type of terminal you are using (as defined in the `TERM` environment variable). This error may also result because the system cannot find the ProvideX library.
- Error #101: *to* Error #102: *No message.*  
Reserved for future use.
- Error #105: Keyed file error (Short key block)  
Reported whenever a key/data block is read from the file and the OS reports that the data read is less than was expected. Normally this indicates a truncated file caused by OS failure.



**Note:** Errors 105 to 117 signal a logical error in the format of the Keyed file being referenced. Try to recover the file if possible, either by restoring it from a backup or by running the ProvideX Keyed/Direct file key reconstruction utility (`*UFAR`).

- Error #106: Keyed file error (Bad key block hdr)  
The key/data block read has an invalid header. The first byte of the data block which indicates the type of data stored within the block is incorrect. See **note** above.
- Error #107: Keyed file error (Wrong key block addr)  
The logical address field within the key/data block (offset 2,4) does not match the address that was expected. See **note** above.
- Error #108: Keyed file error (Record length invalid)  
A data record read from the file has an invalid record length field. The record length *must not* exceed the maximum record length for the file. See **note** above.
- Error #109: Keyed file error (Deleted record chain bad)  
The deleted record chain on a fixed record length Keyed file is corrupted. Each record on this chain should have a deleted record flag set. See **note** above.
- Error #110: Keyed file error (No EOF flag found)  
The key structure has become corrupted as there is no logical EOF key entry. See **note** above.
- Error #111: Keyed file error (Record data unreadable)  
The system was unable to read a record from a fixed record length Keyed file. The operating system is indicating that the data is not available, usually due to file truncation. See **note** above.
- Error #112: Keyed file error (Record key size invalid)  
A data record read from the file has an invalid external key size length. The external key size *must not* exceed the maximum defined for the file. See **note** above.
- Error #113: Keyed file error (Variable record offset bad)  
An offset within a data block is invalid. The offset which indicates where within the data block the physical record starts must contain a positive value not exceeding the size of the data block. See **note** above.
- Error #114: Keyed file error (Physical record address bad)  
A logical record address in a variable length record file is incorrect. A logical address consists of a block address plus a one-byte record index within the block. The record index must be between 1 and 255. This error is reported if the index is zero. See **note** above.
- Error #115: File I/O Verification Error  
Attempt to verify a file **READ** or **WRITE** failed. This error is only reported when using the 'VR' or 'VW' system parameters.
- Error #116: Invalid field descriptor byte  
The contents of a record stored in a file with a dynamic field separator (**SEP=\***) could not be parsed due to an invalid field descriptor or field identifier.



**Error #117: Invalid segment number**

The record address contained within a key block contains an invalid file segment reference. This may be reported when running ProvideX versions prior to 4.23 or when using ProvideX ODBC driver versions prior to 3.22.

**Error #118: Keyed file error**

Decompression failed.

**Error #119: *No message.***

Reserved for future use.

**Error #120: Internal system logic error**

Contact Sage Software Canada Ltd.

**Error #121: Invalid program format**

The Embedded I/O program associated with a Keyed file could not be loaded.

**Error #122: *No message.***

Reserved for future use.

**Warning #123: The following statement labels cannot be located**

This warning appears when a non-existent label is referenced in a program.

**Warning #124: The following statement labels occur more than once**

This warning appears when the same label is defined more than once in a program.

**Warning #125: Improper structure detected**

This warning appears when faulty decision/loop logic is detected in a program.

**Error #126: Forced termination - No valid activation file**

Contact Sage Software Canada Ltd.

**Error #127: Break key depressed**

An internal error generated when either the user hits the **BREAK** key or an **ESCAPE** directive is encountered.

**Error #256 (and >256): *Operating System Errors***

Any error message over 255 is a reported operating system error. ProvideX reports OS errors by taking the actual OS error number and adding 256. Use the following requests to determine what these errors are:

**PRINT MSG(error#) or PRINT MSG(RET)**





# Index

@  
A  
B  
C  
D  
E  
F  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
X  
Y  
Z

## @

- !, exclamation point 25
- 'I9' parameter 697
- 'IB' parameter 697
- 'ID' parameter 698
- 'IF' parameter 698
- 'II' parameter 698
- 'IK' parameter 698
- 'IQ' parameter 698
- 'IR' parameter 698
- 'IS' parameter 699
- 'IT' parameter 699
- 'IU' parameter 699
- 'IW' parameter 699
- 'IX' parameter 700
- ", quotation marks 25
- \$, dollar sign 25
- %, percent sign 25
- ' , back apostrophe 26
- \*, asterisk 25, 252
- \*\*K' parameter 700
- \*\*L' parameter 700
- / or \ slashes (forward or back) 26
- :, colon 26
- : -> ->} prompts 26
- ; semicolon 25
- ? question mark 26
- '@@@' mnemonic 585
- @X( ) / @Y( ) functions, return coordinates 390–391
- ' , apostrophe 25, 823
- '@@@' define cursor position sequence 585
- '1U' parameter 655
- '2D' mnemonic 585
- '3D' mnemonic 586

- '3D' parameter 655
- '4D' mnemonic 586

## A

- 'AB' mnemonic 586
- ABS( ) function, absolute value 392
- accent characters 67, 70, 74
- ACCEPT directive, read single keystroke 28
- ACS( ) function, return arc-cosine 393
- ActiveX, *See* Component Object Model (COM)
- ADD INDEX directive, add key to keyed file 29
- ADDR directive, load/lock program 30
- 'AD' parameter 655
- 'AH' parameter 656
- 'AI' parameter 656
- AND( ) function, logical AND 394
- apostrophe ( ' ) 25
  - back apostrophe ( ' ) 26
  - operator for controls/objects 823
  - parameters 653
- 'AP' parameter 656
- application, *See* program
- arc-cosine, ACS( ) function 393
- arc-sine, ASN( ) function 397
- arc-tangent, ATN( ) function 399
- 'ARC' mnemonic 586
- arguments 19, 40
  - ARG( ) function, in command-line 395
  - CALL directive, transferred to subprogram 40
  - ENTER directive, in subprogram 119
  - NAR system variable, number at startup 567
  - system limits 825
  - See also* parameters, syntax, variables
- arithmetic, *See* math
- arrays
  - 'AD' parameter, auto-DIM 655



- @**
- A**
- B**
- C**
- D**
- E**
- F**
- G**
- H**
- I**
- J**
- K**
- L**
- M**
- N**
- O**
- P**
- Q**
- R**
- S**
- T**
- U**
- V**
- W**
- X**
- Y**
- Z**
- DIM()** function, get array dimensions [415](#)
- DIM** directive, define arrays, strings [86–88](#)
- REDIM** directive, redimension array [277](#)
- system limits [825](#)
- ASCII
- ASC()** function, get ASCII value [396](#)
- ASCII from Radix-40, **TRX()** function [542](#)
- ASCII to Radix-40, **RDX()** function [509](#)
- CHR()** function, ASCII character of value [403](#)
- CMP()** function, compress data [404](#)
- UCP()** function, compress data [547](#)
- ASN()** function, returns arc-sine [397](#)
- assignment operators [821](#)
- asterisk (\*) [25, 252](#)
- ATH()** function, convert hex [398](#)
- '**AT**' mnemonic [587](#)
- ATN()** function, return arc-tangent [399](#)
- attributes, *See* Keyed files, graphical control objects
- Auto Complete [218, 709](#)
- AUTO** directive, automatic line generation [31](#)
- '**AW**' parameter [656](#)
- B**
- '**+B**' & '**-B**' mnemonics [587](#)
- '**BO**' parameter [656](#)
- '**Bn**' mnemonic [588](#)
- back apostrophe (') [26](#)
- '**BACKGR**' mnemonic [588](#)
- base 10 logarithm, **LOG()** function [475](#)
- Basic, *See* language compatibility
- '**BB**' mnemonic [588](#)
- BBx, *See* language compatibility
- '**BEEP**' mnemonic [589](#)
- BEGIN** directive, reset files and variables [32](#)
- behaviour-related mnemonics [581](#)
- '**BE**' mnemonic [589](#)
- '**BF**' parameter [657](#)
- '**BG**' mnemonic [589](#)
- '**BI**' mnemonic [590](#)
- binary [341, 565, 580](#)
- BIN()** function, binary from numeric [400](#)
- DEC()** function, get binary of string [414](#)
- NOT()** function, invert string bits [490](#)
- POS()** function, scan string [502](#)
- SWP()** function, swap data [528](#)
- test for serial files [657](#)
- See also* logic
- bitmaps/icons [36, 49, 179, 192, 203, 267, 346](#)
- \***BITMAP\*** special file [738](#)
- '**PICTURE**' mnemonic, define/draw image [631](#)
- SAVE CONTROL** directive, screen capture [296](#)
- SAVE FILE** directive, save bitmap file [298](#)
- '**BJ**' mnemonic [590](#)
- BKG** system variable, background status [556](#)
- '**BK**' mnemonic [590](#)
- '**BLACK**' & '**\_BLACK**' mnemonics [591](#)
- '**BL**' parameter [657](#)
- '**BLUE**' & '**\_BLUE**' mnemonics [591](#)
- '**BM**' mnemonic [591](#)
- '**BO**' mnemonic [591](#)
- bookmarks in PDF [627, 747](#)
- '**BOX**' mnemonic [592](#)
- branching
- CASE** directive, define branch points [42](#)
- DEFAULT** directive, branch if no **CASE** [77](#)
- END SWITCH** directive, end branching [115](#)
- SWITCH** directive, branch control [331](#)
- BREAK** directive, immediate exit of loop [33](#)
- '**BR**' mnemonic [592](#)
- '**BS**' mnemonic [592](#)
- BSZ()** function, bank memory size [401](#)
- '**BT**' mnemonic [593](#)
- '**BT**' parameter [657](#)
- '**BU**' mnemonic [593](#)
- Business Basic, *See* language compatibility
- BUTTON** directive, control button [34–37](#)
- object properties [703](#)
- '**BW**' mnemonic [593](#)
- '**BX**' mnemonic [593](#)
- '**BX**' parameter [658](#)
- BYE** directive, terminate ProvideX session [39](#)
- '**BY**' parameter [658](#)
- C**
- '\***C**' mnemonic [594](#)
- '**Cn**' mnemonic [594](#)
- Calendar [711](#)
- Calendar button [219](#)
- call
- CALL** directive, transfer to subprogram [40](#)
- program to call on close [649](#)
- STK()** function, stack [522](#)
- XEQ()** function, inline to subprogram [551](#)
- '**CAPTION**' mnemonic [594](#)



- @** CASE directive, define branch points 42
- A** 'CD' parameter 658
- B** 'CE' mnemonic 595
- C** 'CE' parameter 658
- B** 'CF' mnemonic 595
- C** 'CF' parameter 658
- C** channels *xiv*, 22, 826  
*See also* files
- D** character display  
 mnemonics 584  
*See also* text
- E** CHART directive, control chart 43  
 object properties 704
- F** CHECK\_BOX directive, control check box 47,  
 49, 51–53  
 object properties 704
- G** CHG() function, if variable changed 402
- H** 'CH' mnemonic 595
- I** CHN system variable, channels open 556
- J** 'CH' parameter 659
- K** CHR() function, ASCII character of value 403
- L** 'CI' mnemonic 595
- M** 'CI' parameter 659
- N** 'CIRCLE' mnemonic 596
- O** clear  
 BEGIN directive 32  
 CLEAR directive, reset variables 54, 87  
 cursor to EOL 596  
 data from chart 43  
 data from file, PURGE directive 263  
 data from file, REFILE directive 278  
 input type-ahead buffer 595  
 screen 598  
 text from cursor 595
- P** client-server, *See* networks
- Q** CLIP\_BOARD directive, Windows clipboard 55
- R** 'CL' mnemonic 596
- S** CLOSE directive, close file 56  
 output on close 594  
 program to call on close 649
- T** CMP() function, compress data 404
- U** colon (:) 26
- V** 'COLOR' & '\_COLOR' mnemonics 596
- W** 'COLOUR' & '\_COLOUR' mnemonics 596
- X** columns  
 MXC() function, maximum in file 488
- Y** command line  
 ARG() function, return argument 395  
 NAR system variable, argument number 567
- Z** system limits 825
- Command mode 19, 31, 687  
 display errors in 664  
 prompt 19, 681, 683
- commands  
*See* directives
- INVOKE directive, execute OS command 163  
 send OS command string 634  
 special command tags 769–806
- SYS() function, invoke OS command 529
- comments  
 exclamation point (!) 25  
 REM directive, remark 280
- compatibility, *See* language, conversion
- compiled format  
 CPL() function, compile string 407  
 LIST directive, convert statements 176  
 LST() function, convert statements 477  
 PGM() function, return program line 500  
*See* program
- Component Object Model (COM) 823, 825
- DEF OBJECT directive, define object 71
- DELETE OBJECT directive 84
- ON EVENT directive, COM event  
 processing 228
- compress data 404, 547  
*See also* Zlib
- console  
 input/output 175
- CONTINUE directive, next iteration of loop 57
- control object properties 701–735  
 apostrophe operator 823  
 compound properties 728  
 drag and drop (grid) 733  
 loading by row (grid) 733  
 load on demand (list boxes) 729  
 multiple selections (lists and grids) 730  
 multi-property access 728  
 state indicators (tree view list boxes) 731  
*See also* graphical control objects, Graphical  
 User Interface
- control options 810
- conventions  
 in this documentation *xiv*  
 universal naming convention (UNC) 417,  
 421, 757, 761
- conversion  
 accent conversion table 74  
 ASCII to Radix-40, RDX() function 509  
 character sets, TRANSLATE directive 341  
 compiled to readable/list format 176, 477



**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

formatted date, **DTE()** function 422  
Hex to ASCII, **ATH()** function 398  
numeric to string, **STR()** function 525  
Radix-40 to ASCII, **TRX()** function 542  
string to numeric, **NUM()** function 492  
string to other formats, **CVS()** function 412  
tables 340–341, 532, 614, 624  
to/from IEEE, **I3E()** function 456  
X,Y coordinates 390–391  
*See also* language compatibility  
coordinates *xiv*, 390–391, 544–545, 579, 633, 635  
'CO' parameter 659  
**COS()** function, return cosine 406  
cosine  
  **ACS()** function, return arc-cosine 393  
  **COS()** function, return cosine 406  
'CPI' mnemonic 597  
**CPL()** function, compile string 407  
'CP' mnemonic 597  
**CRC()** function, cyclic-redundancy-check 408  
**CREATE TABLE** directive, create EFF file 58  
'CR' mnemonic 598  
**CSE()** function 409  
**CSE()** function, case compare 409  
'CS' mnemonic 598  
'CS' parameter 660  
CTL  
  **CTL()** function, return CTL definition 410  
  CTL system variable, end input code 557  
  **DEFCTL** directive, define CTL values 78  
  negative CTL definitions 817  
  **PREINPUT** directive, prime input queue 254  
  **SET\_FOCUS** directive, set input focus 304  
  **SETCTL** directive, GOSUB on CTL event 307  
  *See also* graphical control objects  
'CT' parameter 660  
*ctrlopt*, control options 810  
CUI (Character User Interface)  
  *See* character display, text mode  
'CU' parameter 660  
cursor 592, 594–596, 599, 602  
  '**CURSOR**' mnemonic 598  
  motion mnemonics 582  
  *See also* prompts, mouse  
**CUSTOM\_VBX** directive, create/control VBX 61  
**CVS()** function, convert string 412  
**CWDIR** directive, change working directory 62  
'CYAN' & '\_CYAN' mnemonics 599  
cyclic-redundancy-check, **CRC()** function 408

**D**

'+D' & '-D' mnemonics 599  
'D0' parameter 660  
data  
  **DATA** directive, define data elements 63  
  **DICTIONARY** directive, data dictionary 85  
  **EXTRACT** directive, read and lock data 126  
  **FIND** directive, locate and read data 131  
  **SWP()** function, swap data 528  
  *See* files, records, variables  
date  
  buffers 658  
  date table 67, 70, 74  
  **DAY\_FORMAT** directive, format for **DAY** 64  
  **DAY** system variable, current date 557  
  define date table 67, 70, 74  
  **DTE()** function, convert date 422  
  **JUL()** function, Julian date 463  
  MAS 90 date format 697  
  **SETDAY** directive, change local date 308  
[**DB2**] special command tag 770–774  
'DB' parameter 660  
'DC' mnemonic 599  
'DC' parameter 661  
[**DDE**] special command tag 776  
'DD' parameter 661  
debugging 122, 317, 468  
  *See also* error handling  
**DEC()** function, get binary of string 414  
decryption value, **HSK()** function 451  
**DEFAULT** directive, branch if no case 77  
'DEFAULT' mnemonic 600  
**DEF CLASS** directive, define object class 65  
**DEF CTL** directive, define control signal 76  
**DEFCTL** directive, define CTL values 78  
**DEF CVS** directive, define accent table 74  
**DEF DTE** directive, define date table 74  
**DEF EOM** directive, define EOM character 76  
**DEF ERR** directive, define system error value 76  
**DEF FN** directive, define function 68  
**DEF GID** directive, define UNIX group ID 67  
  definiton mnemonics 581  
**DEF LCS** directive, define lowercase table 74  
**DEF LFA** directive, define last file number  
  accessed 76  
**DEF LFO** directive, define last file number  
  opened 76  
**DEF MSG** directive, define temporary  
  message 70



**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

- DEF OBJECT** directive, define COM 71
- DEFPRN** directive, define as printer 81
- DEF RET** directive, define OS error code 76
- DEFTTY** directive, define terminal size 82
- DEF UCS** directive, define uppercase table 74
- DEF UID** directive, define UNIX user ID 67
- delete
  - character at cursor 599
  - DELETE** directive, remove program lines 83
  - DELETE OBJECT** directive, drop COM Object 84
  - DROP CLASS** directive, class definition 102
  - DROP OBJECT** directive 104, 382
  - ERASE** directive, file or directory 120
  - object in scroll region 602
  - REMOVE** directive, record from file 281
- device
  - control sequences (mnemonics) 577–651
  - ProvideX devices 737–767
  - SETDEV** directive, set device type name 309
  - time-out 662
  - See terminals, printers
- '**DF**' parameter 661
- '**DIALOGUE**' mnemonic 600
- DICTIONARY** directive, data dictionary 85
- DIM( )** function, fill string/get array size 415
- DIM** directive, define arrays and strings 86
- DIR( )** function, get current directory 417
- Direct, Keyed file type 22
  - DIRECT** directive, create Direct file 89
- directives 27–387
  - concepts 19
  - conventions regarding syntax *xiv*
- directory
  - change current, **CWDIR** directive 62
  - check before checking prefix 658
  - delete from system, **ERASE** directive 120
  - delimiter, **DLM** system variable 558
  - DOS delimiter 661
  - get current, **DIR( )** function 417
  - home/starting, **HWD** system variable 563
  - path of current, **LWD** system variable 565
  - subdirectory delimiter 686
- DIRECTORY** directive, create subdirectory 91
- dirty file indicator 337
- DISABLE CONTROL** directive 93
- DISABLE** directive, disable prefix 92
- DISABLE EVENT** directive, Internal Event Disable 94
- disk 92, 110, 249, 417, 434
- CWDIR** directive, change directory/drive 62
- DSK( )** function, get current disk drive 421
  - input/output 237, 305
  - manage disk space 120, 263, 278
  - pathname 438, 441
- SETDRIVE** directive, change default drive 315
- display
  - control window display 639
  - OS errors in command mode 664
  - popup message, **MSGBOX** directive 212
  - printable data, **PRINT** directive 255
  - variables, **DUMP** directive 106
- [**DLL**] special command tag 778, 781
- DLL( ) / DLX( )** functions, call DLL 418
- DLM** system variable, directory delimiter 558
- '**DL**' parameter 662
- '**DN**' mnemonic 602
- dollar sign (\$) 25
- '**DO**' mnemonic 602
- DOS 372, 401, 567, 665, 686, 694, 769
  - delimiter 558, 661
  - DIR( )** function, get current directory 417
  - DOS-only mnemonics 588, 603, 648, 650
  - DOS-only parameters 665, 675, 677, 679, 681, 692, 697
  - DSK( )** function, get current disk drive 421
  - line mode 650
  - wide printer mode 648
- '**DP**' parameter 662
- drive, See disk, devices, directory
- DROP.ON** directive, drag and drop 105
- DROP\_BOX** directive, control drop box 96–101
  - drop box write error 623
  - object properties 704
  - VARDROP\_BOX** directive 354
- DROP CLASS** directive, drop class definition 102
- DROP** directive, unload program 95
- DROP INDEX** directive, drop key from file 103
- '**DROP**' mnemonic 602
- DROP OBJECT** directive, delete object 104, 382
- DSK( )** function, get current disk drive 421
- DSZ** system variable, data space available 559
- DTE( )** function, convert date 422
- '**DT**' parameter 662
- DUMP** directive, display variables 106
- '**DW**' parameter 663



**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

## E

'+E' & '-E' mnemonics 603

'EB' mnemonic 603

echoing 28, 227, 589, 598, 603

editing

**EDIT** directive, edit line in program 108

    mnemonics related to 582

'EE' mnemonic 603

EFF (Enhanced File Format) 22, 59, 167, 334,  
671, 683

'EF' mnemonic 603

'EG' mnemonic 603

'EG' parameter 663

'EI' mnemonic 604

'EJ' mnemonic 604

'EL' mnemonic 604

'EL' mnemonic 604

'EL' parameter 663

'EM' mnemonic 605

emulation, *See* language compatibility

**ENABLE CONTROL** directive, enable CTL 111

**ENABLE** directive, re-enable prefix 110

**ENABLE EVENT** directive, Internal Event  
    Enable 112

encryption

    level, 'EL' parameter 663

    option with **SAVE** directive 295

**PASSWORD** directive 239

encryption value, **HSH()** function 451

**END\_IF** directive, end IF directive 117

**END\_DEF** directive, end function definition 114

**END** directive, halt program execution 113

*See also* terminate

**END SWITCH** directive, end branching 115

**ENDTRACE** directive, end trace output 118

**END WITH** directive 116

Enhanced File Format (EFF) 22, 59, 167, 334,  
671, 683

**ENTER** directive, specify arguments 119

entry point 26

**ENV()** function, get environment values 424

'EO' mnemonic 605

**EOM** system variable, EOM string 559

'EO' parameter 664

'EP' mnemonic 605

**EPT()** function, return exponent value 426

**ERASE** directive, delete file/directory 120

'ER' mnemonic 605

**ERR()** function, test error value 427

error

    handling in ProvideX 24

    codes and messages 828–841

    drop box write error 623

**ERR()** function, test error value 427

**ERR** system variable, system-detected 560

**ERS** system variable, line number of last 560

    invalid mnemonic 589, 603

**MSG()** function, message text 484

    OS errors in command mode 664

**RET** system variable, last OS error code 571

**SETERR** directive, set error transfer 316

**ERROR\_HANDLER** directive, assign generic error  
    handler 121

**ERR** system variable, last detected error 560

**ERS** system variable, error's line number 560

**ESCAPE** directive, interrupt execution 122

**ESC** system variable, escape character 561

'ES' mnemonic 606

'ES' parameter 664

'ET' mnemonic 606

'EU' mnemonic 606, 618, 633

Event Handling

    \***SYSTEM** 751

Event Handling 71, 228, 805, 823, 825

**DISABLE EVENT** directive 94

**ENABLE EVENT** directive 112

*See also* Component Object Model (COM)

**EVN()** function, evaluate numeric 429

**EVS()** function, evaluate string 430

'EW' mnemonic 606

exclamation point (!) 25, 280

**EXECUTE** directive, execute command 123

Execution mode 19

exit

    from a loop, **BREAK** directive 33

    from stack, **POP** directive 245

    return CTL=4 on exit 665

    to OS on end of program 697

**EXIT** directive, halt subprogram and return 124

**EXITTO** directive, end loop, transfer control 125

**EXP()** function, raise to base ten 431

exponent

    assignment, arithmetic operators 821

**EPT()** function, return power of 10 426

'EX' parameter 664

expressions, syntax conventions of *xiv*

'!V' parameter 699

**EXTRACT** directive, read data 126





**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

**EXTRACT RECORD** directive, read record 128

## F

'+F' & '-F' mnemonics 606

'F,' parameter 665

'Fn' mnemonic 607

'F4' parameter 665

'FB' parameter 665

'FC' parameter 665

'FE' parameter 665

'FF' mnemonic 607

**FFN( )** function, find file number 432

'FF' parameter 666

**FIB( )** function, file information block 434

**FID( )** function, file information descriptor 438

**FID(0 )** definition 320

**FI** directive, *See* **END\_IF**

fields

extended field attributes, **XFA( )** function 552

field separator 515

*fileopt*, file options 810

files 22

'FF' parameter, file format 666

create/assign program file 259

**DIRECT** directive, create Direct file 89

dirty file indicator 337

dynamic buffering 660

EFF files 22

**FFN( )** function, find file number 432

**FIB( )** function 434

**FID( )** function 438

**FILE** directive, new file from FID, FIB 130

**FIN( )** function 441

flushing 587

**GET\_FILE\_BOX** directive, filename entry 139

**INDEXED** directive, create Indexed file 159

input/output options 810

**KEYED** directive, create Keyed file 58, 166

KNO (file access key) 811

**LOCK** directive, reserve file 200

logging 334, 337

logical file numbers 22

markup files 591

**MXC( )** function, maximum columns in 488

**MXL( )** function, maximum lines in 488

**OPEN** directive, open files for processing 232

output buffering on/off 587

print files 752

**PROGRAM** directive, create program file 259

**PURGE** directive, clear data from file 263

**READ** directive, data from file 271

**READ RECORD** directive, record from file 275

**RENAME** directive, change a file's name 282  
search rules 249

**SERIAL** directive, create a sequential file 302

**SORT** directive, create file for sorting 327

special files 737-767

system limits 826

**UNLOCK** directive, remove file lock 349

VLR files 22

*See also* Keyed, Index, and Program files

'**FILL**' mnemonic 607

**FIN( )** function, return file information 441

**FIND** directive, locate data 131

**FIND RECORD** directive, locate data record 132

'FI' parameter 666

'FL' mnemonic 608

**FLOATING POINT** directive, switch to scientific notation 133

'FL' parameter 666

FLR, (fixed length record) 22

'FN' parameter 667

focus

change of focus on/off 606

**SET\_FOCUS** directive 304

*See* graphical control objects

'**FONT**' mnemonic 609

fonts 626, 811

'FO' parameter 667

**FOR..NEXT** directive, incremented loop 134

format

**DAY\_FORMAT** directive 64

masks 160, 255, 525, 813

*See* syntax, dates, control object properties

formfeed, *See* printers

'FP' parameter 667

**FPT( )** function, return fractional part 445

fractional parts, **FPT( )** function 445

'**FRAME**' mnemonic 610

'FS' parameter 667

'FT' parameter 668

**FUNCTION** directive, declare object method 137

functions 389-554

**DEF FN** directive, define function 68

**END DEF** directive, end definition of 114

function keys 608

'FU' parameter 668

'FX' parameter 668



**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

**G**

**GAP()** function, return odd parity string 446  
**GBL()** function, global string variable 447  
 'GD' mnemonic 611  
 'GE' mnemonic 611  
**GEP()** function, return even parity string 449  
**GET\_FILE\_BOX** directive, ask for filename 139  
 'GF' mnemonic 612  
**GFN** system variable, global file number 561  
**GID** system variable, OS process ID 562  
 global  
   **GBL()** function, global string variable 447  
   **GFN** system variable, global file number 561  
   global variables 25  
   search utility 25  
**GOSUB..** directive, execute subroutine 141  
**GOTO** directive, transfer within program 142  
 'GOTO' mnemonic 612  
 gradient fill 607  
 graphical control objects 21, 703–708  
   2D, 3D, and 4D controls 585–586  
   **BUTTON** directive 34–37  
   **CHECK\_BOX** directive 47, 49, 51–53  
   **DISABLE CONTROL** directive 93  
   **DROP\_BOX** directive 96–101  
   **ENABLE CONTROL** directive 111  
   GDI resources 21  
   **GRID** directive 143–152  
   **H\_SCROLLBAR** directive 153–155  
   **HIDE** directive 156  
   **LIST\_BOX** directive 178–188  
   list view list boxes 189  
   **MENU\_BAR** directive 202–203, 205  
   **MULTI\_LINE** directive 215, 217, 222  
   **POPUP\_MENU** directive 246  
   **RADIO\_BUTTON** directive 265–269  
   redisplay hidden control 326  
   **RENAME CONTROL** directive 284  
   return object information 493  
   **SHOW** directive 326  
   tree view list boxes 192  
   **TRISTATE\_BOX** directive 344, 346–348  
   **V\_SCROLLBAR** directive (vertical) 365–367  
   **VARDROP\_BOX** directive 354–356, 358–359  
   **VARLIST\_BOX** directive 360–364  
   *See also* control object properties, Graphical User Interface  
 Graphical User Interface (GUI) 21, 579  
   caption for window 594  
   control window display 639

mnemonics 582–583  
 size of window 579  
 window number 455  
 window region 579  
 X,Y coordinates 579  
*See also* graphical control objects  
 graphics  
   \***BITMAP\*** special file 738  
   bitmaps/icons 36, 49, 179, 192, 203, 267, 346  
   drawing frame, box, line, arc, polygon, rectangle, circle 579  
   '**IMAGE**' mnemonic, define graphics group 615  
   '**PICTURE**' mnemonic, define/draw image 631  
   **SAVE CONTROL** directive, screen capture 296  
   **SAVE FILE** directive, save bitmap file 298  
   '**GREEN**' & '**\_GREEN**' mnemonics 612  
**GRID** directive, control grid 143–152  
   drag and drop 733  
   loading and accessing 733  
   object properties 705  
 '**GS**' mnemonic 613

**H**

'\*H' mnemonic 613  
**H\_SCROLLBAR** directive, control horizontal scrollbar 153  
   object properties 708  
 halt, *See* terminate  
 hash value, **HSH()** function 451  
 'HC' parameter 668  
 help  
   message identifier 160, 227  
   restore default 202  
   **SYSTEM\_HELP** directive, invoke help 332  
 hexadecimal  
   **ATH()** function, convert hex 398  
   **HTA()** function, hex value of string 454  
   values in ProvideX 25  
**HFN** system variable, highest channel 562  
**HIDE** directive, hide control object 156  
 'HIDE' mnemonic 639  
**HLP** system variable, last specified **HLP=** 563  
 'HP' parameter 669  
**HSA()** function, highest sector available 450  
**HSH()** function, generate modified value 451  
**HTA()** function, get hex value of string 454



- @** \*HTML\* 740  
**A** HWD system variable, home directory 563  
 HWN( ) function, highest window number 455
- B**
- C**
- D**
- E**
- F**
- G**
- H**
- I**
- J**
- K**
- L**
- M**
- N**
- O**
- P**
- Q**
- R**
- S**
- T**
- U**
- V**
- W**
- X**
- Y**
- Z**
- \*HTML\* 740  
 HWD system variable, home directory 563  
 HWN( ) function, highest window number 455
- '\*' mnemonic 614  
 '+!' & '-!' mnemonics 614  
 'I0' parameter 669  
 'I2' parameter 669  
 I3E( ) function, convert to/from IEEE 456  
 'IC' mnemonic 614  
 icons  
   dialogue icon 626  
 'IC' parameter 669  
 IEEE, I3E( ) function 456  
 IF ... THEN ... ELSE directive, test condition 157  
 'IMAGE' mnemonic 615  
 'IM' parameter 669  
 IND( ) function, return next record index 457  
 Indexed, file type 22  
   INDEXED directive, create Indexed file 159  
 INPUT directive, get input from terminal 160  
 INSERT directive, insert new record in file 162  
 instructions, *See* directives, statements  
 INT( ) function, return integer portion 458  
 interrupt  
   ESCAPE directive, suspend execution 122  
   *See also* exit, terminate  
   SETESC directive, handler program 317  
 INVOKE directive, execute OS command 163  
 IOList  
   IOL( ) function, get specification 459  
   IOLIST directive, specify variable list 165  
   REC( ) function, expand specification 510  
 IOR( ) function, OR comparison 460  
 'IR' parameter 670  
 'IS' parameter 670  
 'IW' parameter 670  
 'IZ' parameter 670
- J**  
 JavX *xiii*
- 'JC' mnemonic 616  
 'JC' parameter 670  
 'JD' mnemonic 616  
 'JL' mnemonic 616  
 'JN' mnemonic 616
- journalization, SYSTEM\_JRNL directive 334  
 'JR' mnemonic 616  
 'JS' mnemonic 616  
 JST( ) function, justify string 461  
 JUL( ) function, return Julian date 463
- K**
- KEC( ) function, key of current record 465  
 KEF( ) function, return first key of file 466  
 KEL( ) function, return last key of file 467  
 KEN( ) function, return key after next 468  
 KEP( ) function, return prior record's key 469  
 KEY( ) function, return key of next record 470  
 KEYED directive, create Keyed file 166  
 Keyed files 22, 671  
   ADD INDEX directive, add key to file 29  
   automatic padding of 272, 383, 386  
   BBx emulation 671  
   buffers 677  
   DIRECT directive, file with keyed access 89  
   DROP INDEX directive, drop from file 103  
   I/O buffers 657  
   KEF( ) function, first key of file 466  
   KEL( ), last key of file 467  
   KEN( ), key after next 468  
   KEP( ), prior record's key 469  
   KEY( ) function, key of next record 470  
   Key definition attributes 167  
   KEYED directive 58, 166  
   KGN( ) function, generate record key 471  
   opening static Keyed file 237  
   'QK' parameter 683  
   RENAME..INDEX directive, rename keys 285  
   SETDEV KEY directive, change keys 311  
   shrink files 687  
   special for paths 250  
   system limits 826  
   writing to 384  
 KEYED LOAD directive, rebuild keyed file 172  
 KGN( ) function, generate record key 471  
 KNO (file access key) 811  
 'KF' parameter 671  
 'KR' parameter 671
- L**
- 'L6' mnemonic 617  
 'L8' mnemonic 617  
 labels, logical statement references 816



- @** language compatibility  
 BBx 79, 88, 440, 643, 651, 658, 661, 666, 671, 691, 700
- A** **BSZ( )** function 401
- B** conversion to ProvideX 17, 352, 521, 604, 608, 641, 664, 670, 822
- C** dates 423
- D** **DISABLE** directive 92
- E** emulation 79, 643, 658, 661, 666, 671, 690–691
- F** **ENABLE** directive 110
- G** **GBL( )** function 447
- H** **HSA( )** function 450
- I** Rexon 666
- J** **SETDRIVE** directive, change default drive 315
- K** **SHORT\_FORM** directive 325
- L** **SSZ( )** function 521
- M** Thoroughbred 487, 666, 690, 822
- N** **TSM** system variable 575
- O** **XFA( )** function 552
- P** *See also* ProvideX
- Q** **LAOD**, *See* **LOAD** directive 194
- R** Large File Support (LFS) 59, 167, 671
- S** **'LB'** parameter 671
- T** **'LC'** mnemonic 617
- U** **'LC'** parameter 671
- V** **LCS( )** function, return lowercase string 74, 472
- W** **'LD'** mnemonic 617
- X** **'LD'** parameter 672
- Y** **LEN( )** function, return string length 473
- Z** **'LE'** parameter 672
- 'LET'** directive, assign value to variable 173
- lexicon 352
- See also* language compatibility
- LFA** system variable, last file accessed 563
- 'LF'** mnemonic 617
- LFO** system variable, last file opened 564
- 'LF'** parameter 672
- LFS (Large File Support) 59, 167, 671
- Libharu 538, 669, 746
- library
- DLL( ) / DLX( )** 418
  - message library 208, 484
  - PDF forms library 748
  - program library 781
  - XML Library 764
- LIKE** directive, inherit properties 174
- limits preset in ProvideX 825
- 'LI'** mnemonic 617
- lines
- 'LINE'** mnemonic 618
- AUTO** directive, automatic line generation 31
- labels 26
- LINE\_SWITCH** directive, switch I/O 175
- MXL( )** function, maximum in file 488
- RENUMBER** directive, renumber lines 286
- system limits 825
- See also* statements
- LIP** system variable, last input location 564
- LIST\_BOX** directive, control list box 178–188
- list view list boxes 189
  - load on demand 729
  - object properties 705
  - tree view list boxes 192
- LIST** directive, list program statements 176
- list view list boxes 189
- load on demand 729
  - object properties 706
  - row highlighting 645
- 'LM'** parameter 672
- LNO( )** function, return line number 474
- LOAD CLASS** directive, pre-load class definition 195
- LOAD DATA** directive, load program constants 196
- LOAD** directive, load program into memory 194
- load on demand (list boxes) 729
- LOCAL** directive, designation of local data 197
- LOCK** directive, reserve file 200
- LOG( )** function, return base 10 logarithm 475
- logic 105, 502
- AND( )** function, AND comparison 394
  - descending key 698
  - IOR( )** function, OR comparison 460
  - logical ON/OFF control 38, 52
  - logical statement references (labels) 816
  - NOT( )** function, condition 490
  - ON\_CREATE/ON\_DELETE definitions 489
  - pre-display (NOMADS) 256
  - XOR( )** function, exclusive OR 554
- See also* branching, program, Object Oriented Programming
- LONG\_FORM** directive, use long names 201
- longitudinal-redundancy check, **LRC( )** function 476
- loops
- BREAK** directive, immediate exit of loop 33
  - CONTINUE** directive, next iteration of loop 57
  - EXITTO** directive, exit/transfer from loop 125
  - FOR..NEXT** directives, increment loop 134



- @**
- A**
- B**
- C**
- D**
- E**
- F**
- G**
- H**
- I**
- J**
- K**
- L**
- M**
- MAGENTA** & **'\_MAGENTA'** mnemonics 619
- MAS 90 697, 773, 784, 787, 795
- masks
- data format masks 160, 255, 525, 813
  - password mask character 681
  - scan string for mask 486
  - set date format 64
- math
- arithmetic operators 821
  - ABS()** function, absolute value 392
  - ACS()** function, return arc-cosine 393
  - ASN()** function, returns arc-sine 397
  - ATN()** function, return arc-tangent 399
  - COS()** function, return cosine 406
  - EPT()** function, return exponent value 426
  - EVN()** function, evaluate expression 429
  - EXP()** function, raise to base ten 431
  - LOG()** function, return base 10 logarithm 475
  - MOD()** function, return modulus 483
  - PRC()** function, round to precision 503
  - SGN()** function, return sign of value 516
  - SIN()** function, sine 517
  - SQR()** function, square root 518
  - TAN()** function, tangent 531
- MAX()** function, return maximum value 478
- 'MAXSIZE'** & **'MINSIZE'** mnemonics 619
- 'MB'** parameter 674
- MCI, Multimedia Control Interface 223
- 'MC'** parameter 674
- 'ME'** mnemonic 620
- memory 670, 675, 688, 692, 697
- \*MEMORY\*** 741
  - BSZ()** function, bank memory size 401
  - MEM()** function, return memory value 479
  - system limit 825
- MENU\_BAR** directive, define menu
- bar 202–203, 205
- MERGE** directive, read lines from file 206
- MESSAGE\_LIB** directive, message library 208
- 'MESSAGE'** mnemonic 620
- messages
- DEF MSG** directive, define temporary message 70
  - library files 208
  - message bar text 620
  - return error messages 484
  - See also* error codes and messages
- 'MF'** parameter 675
- MID()** function, return substring 480
- MIN()** function, minimum value in list 481
- 'MINSIZE'** mnemonic 621
- mnemonics 20, 210, 577–651
- overview 577
  - MNEMONIC** directive, define mnemonic 210
  - MNM()** function, mnemonic value 482
  - categories 581
  - DOS-only mnemonics 588, 603, 648, 650
  - dynamic information 580
  - invalid mnemonic 603
- 'MN'** mnemonic 622
- MOD()** function, return modulus 483
- 'MODE'** mnemonic 622
- modes in ProvideX (Execution, Command) 19
- mouse 582
- 'CURSOR'** mnemonic 598
  - define signal 630
  - SETMOUSE** directive, control/set mouse 321
- 'MOVE'** mnemonic 623
- 'MP'** mnemonic 623
- 'MP'** parameter 675
- MSE** system variable, mouse state 565
- MSG()** function, return message text 484
- MSGBOX** directive, popup message box 212
- See also* **POPUP\_MENU** directive
- MSK()** function, scan string for mask 486
- MSL** system variable, mask string length 567
- 'MS'** mnemonic 623
- 'MS'** parameter 675
- 'Z'**



- @** **MULTI\_LINE** directive, control multi-line input 215, 217, 222  
 object properties 706
- A** **MULTI\_MEDIA** directive, control interface 223
- B** **MXC()** / **MXL()** function, return maximum column/line 488
- C** 'MX' parameter 675
- D** **[MYSQL]** special command tag 783–785
- E**
- F** **N**
- G** '+N' & '-N' mnemonics 623
- H** **NAR** system variable, startup arguments 567
- I** navigation tips *xiv*
- J** 'NE' parameter 676
- K** networks 440, 576, 678, 687
- L** **[RPC]** special command tag 797–798  
**[TCP]** special command tag 799  
**[WDX]** special command tag 801  
**NID** system variable, network ID 567  
**PROCESS SERVER** directive 258  
**UID** system variable, current user ID 575
- M** **NEW()** function, create new object 489
- N** **NEXT** directive, end of FOR loop 225  
**NEXT RECORD** directive, end of SELECT 226  
**NID** system variable, network ID 567
- O** 'NI' mnemonic 624  
 'NI' parameter 676  
 'NK' parameter 676  
 'NL' parameter 676  
 'NN' parameter 676  
**NOMADS** *xiii*, 21, 256, 698  
**NOT()** function, logical condition 490  
 'NR' parameter 676  
 'NS' parameter 677  
**NUL()** function, test for null 491  
**NUM()** function, convert string to numeric 492  
 numeric values  
   binary from numeric, **BIN()** function 400  
   contents of variable, **VIN()** function 549  
   convert from string, **NUM()** function 492  
   evaluate expression, **EVN()** function 429  
   maximum value in list, **MAX()** function 478  
   minimum value in list, **MIN()** function 481  
   numeric to string, **STR()** function 525  
   pack numeric data, **PCK()** function 498  
   return random number, **RND()** function 513  
   sign of value, **SGN()** function 516  
   unpack numeric data, **UPK()** function 548  
   See also math
- 'NX' parameter 677
- O**
- '\*O' mnemonic 624
- OBJ()** function, return object information 493  
 object code 259, 352, 459, 510, 688  
 Object Oriented Programming (OOP) 22  
   **DEF CLASS** directive, define object class 65  
   **DROP CLASS** directive, delete class 102  
   **DROP OBJECT** directive, delete object 104, 382  
   **FUNCTION** directive, declare method 137  
   **LIKE** directive, inherit properties 174  
   **LOAD CLASS** directive, pre-load class 195  
   **LOCAL** directive, assign local properties 197  
   methods and properties 65  
   **NEW()** function, create new object 489  
   **OPEN OBJECT** directive 232  
   **PROGRAM** directive, assign program 259  
   **PROPERTY** directive, declare properties 261  
   **RENAME CLASS** directive, rename class 283  
   **STATIC** directive, add local properties at runtime 329  
   WinDX support 805
- objects  
   See Component Object Model (COM)  
   See graphical control objects  
   See Object Oriented Programming (OOP)
- OBTAIN** directive, hidden terminal input 227
- [OCI]** special command tag 786  
   See Oracle Call Interface
- 'OC' parameter 677
- OCX (Object Component eXtension)  
   See Component Object Model (COM)
- [ODB]** special command tag 788, 791, 795–796
- ODBC, See Open DataBase Connectivity
- 'OFFSET' mnemonic 629
- 'OF' parameter 677
- OLE (Object Linking and Embedding)  
   See Component Object Model (COM)
- 'OL' parameter 677
- 'OM' parameter 678
- ON ... GOSUB** directive, conditional subroutine execution 230
- ON ... GOTO** directive, conditional transfer 231
- ON EVENT** directive, COM event processing 228
- OOP  
   See Object Oriented Programming



**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

Open DataBase Connectivity (ODBC) *xiii*, 162, 351, 467–469, 478, 481, 514, 788, 791, 795–796

[ODB] special command tag 791

ODBC to MAS 90 697

SQL display, 'IQ' parameter 698

**OPEN** directive, open a file/device 232

commit updates before 677

file open options 233

maximum buffers for **OPEN LOAD** 677

**OPT( )** function, return file open options 495

**PASSWORD** directive, password-encrypt 239

**PREFIX** directive, define search path 249

special command tags 769

See files, devices

operating system

expanded path, 'OR' parameter 678

**GID** system variable, OS process ID 562

**INVOKE** directive, OS command 163

**RET** system variable, last OS error code 571

**SYS( )** function, invoke OS command 529

**SYS** system variable, OS identification 573

See also Windows, UNIX, DOS, pathname

operators

apostrophe operator 823

arithmetic operators 821

assignment operators 822

conventions in this documentation *xiv*

punctuation/syntax symbols 25

'OP' parameter 678

**OPT( )** function, return file open options 495

options

'OPTION' mnemonic 624

file open options 233, 495

input/output and control options 810

Oracle Call Interface (OCI) 162, 351, 786

[OCI] special command tag 786

OR comparison, **IOR( )** function 460

'OR' parameter 678

output, See mnemonics, **DUMP**, **LIST**, **PRINT**

'OW' parameter 678

**P**

'+P' & '-P' mnemonics 630

**PAD( )** function, pad/truncate string 496

parameters 19

**PRM( )** function, return parameter value 504

**PRM** system variable, list of parameters 570

**SET\_PARAM** directive, set parameters 306

system parameters 653–700

parity value of strings 446, 449

**PASSWORD** directive, password-encrypt 239

pathname

**DIR( )** function, get current directory 417

**PREFIX** directive, set file search rules 249

**PTH( )** function 506

special command tags 769–806

UNC-style 417, 421, 434, 757, 761

See also disk, directory, file

**PCK( )** function, pack numeric data 498

'PC' parameter 678

PDF, Portable Document Format 744

\*PDF\* 744

bookmarks 627, 747

redirecting printer output to PDF 656, 750

Libharu 538, 669, 746

'PD' parameter 679

'PE' mnemonic 630

'PEN' mnemonic 630

'PE' parameter 679

percent sign (%) 25

**PERFORM** directive, transfer to subprogram, share variables 243

'PF' parameter 679

**PFX( )** function, return prefix value 499

**PFX** system variable, current prefix 568

**PGM( )** function, return program line 500

**PGN** system variable, program pathname 568

'PICTURE' mnemonic 631

'PIE' mnemonic 632

'PL' parameter 679

'POLYGON' mnemonic 633

'PO' parameter 680

**POP** directive, premature exit from stack 245

'POP' mnemonic 633

**POPUP\_MENU** directive, popup menu 246

**POS( )** function, scan string 502

'PP' parameter 680

'PQ' parameter 680

**PRC( )** function, round number to precision 503  
precision

**PRC** system variable, current precision 569

**PRECISION** directive, change precision 248

round to precision, **PRC( )** function 503

prefix

**DISABLE** directive 92

**ENABLE** directive 110

**PFX( )** function, return current prefix 499

**PFX** system variable, current prefix 568



**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

- PREFIX** directive, file search rules 249  
table 92, 110, 250
  - PREINPUT** directive, prime input queue 254
  - PRINT** directive, display information 255
  - printers
    - \***VIEWER\*** device file 752, 755
    - \***WINDEV\*** device file 756–758
    - \***WINPRT\*** device file 629, 760–763
    - changing source tray 629
    - defined on servers 376
    - DEFPRT** directive, define as printer 81
    - formfeeds 607
    - maximum columns, lines 488
    - mnemonics 582–584
    - orientation 629
    - print files 752
    - queues 763
    - raw print mode 756
    - set printer default 687
    - spooler 586, 594, 662, 684, 752
    - Windows printer 376, 756–758, 760–763
  - PRM( )** function, return parameter value 504
  - PRM** system variable, ProvideX parameters 570
  - PROCESS** directive, call a NOMADS panel 256
  - PROCESS SERVER** directive, remote server 258
  - program
    - concepts, syntax 19
    - ADDR** directive, load/lock program 30
    - call program on close 649
    - DROP** directive, unload program 95
    - EDIT** directive, edit program 108
    - Embedded I/O program 312
    - END** directive, halt execution 113
    - error status 575
    - library 781
    - LOAD** directive, load into memory 194
    - LPG** system variable, lead program name 564
    - PROGRAM** directive, create program file 259
    - PSZ** system variable, current size 570
    - public programs 507
    - RESET** directive, reset program state 288
    - RUN** directive, transfer/execute program 294
    - SAVE** directive, write to file 295
    - STK( )** function, program call stack 522
    - system limits 825
    - trace 324
    - See also ProvideX, subprogram, compiled format, Object Oriented Programming
  - prompts 19, 26, 681, 683
    - for password 239, 680
    - issued to terminal devices 160, 227
  - properties
    - See control object properties
    - See also Object Oriented Programming
  - PROPERTY** directive, declare properties 261
  - ProvideX
    - devices 737–767
    - environments 19
    - Execution mode/Command mode 19
    - introduction 17–25
    - parameter settings, **PRM** system variable 570
    - restart 328
    - SBB 688
    - search rules 251
    - session 19
    - syntax *xiv*
    - system limits 825
    - terminate session 39, 264, 279
    - utilities 25
    - See also PVX, compiled format, language compatibility
  - '**PS**' mnemonic 634
  - '**PS**' parameter 681
  - PSZ** system variable, current program size 570
  - PTH( )** function, return pathname 506
  - '**PT**' parameter 681
  - PUB( )** function, list public programs 507
  - punctuation 25
  - '**PU**' parameter 681
  - PURGE** directive, clear data from a file 263
  - '**PUSH**' mnemonic 634
  - PVX
    - PVXFID0 environment variable 320, 435, 438
    - www.pvx.com 22
  - '**PW**' parameter 681
  - '**PZ**' parameter 681
- ## Q
- '**Q ^**' parameter 682
  - '**Q \_**' parameter 682
  - '**QD**' parameter 682
  - '**QF**' parameter 683
  - '**QS**' parameter 683
  - '**QK**' parameter 683
  - '**QT**' parameter 683
  - question mark (?) 26
  - QUIT** directive, terminate ProvideX 264
  - quote characters
    - " , double 25
    - ' , single (apostrophe) 25





@  
A  
B  
C  
D  
E  
F  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
X  
Y  
Z

**QUO** system variable, ASCII quote character 571

## R

'\*R' mnemonic 634

**RADIO\_BUTTON** directive, control radio button 265–269

object properties 707

Radix-40

from ASCII, **RDX( )** function 509

to ASCII, **TRX( )** function 542

**RANDOMIZE** directive, set random key 270

random numbers, **RND( )** function 513

'RB' mnemonic 634

**RCD( )** function, return next record 508

'RC' mnemonic 635

**RDX( )** function, ASCII to Radix-40 509

**READ DATA** directive, read from program 273

**READ** directive, read from file 271

**READ RECORD** directive, read record 275

**REC( )** function, expand IOList 510

records

**EXTRACT RECORD** directive, read-lock 128

**FIND RECORD** directive, locate/read 132

**IND( )** function, next record index 457

**INSERT** directive, new record in file 162

**KEC( )** function, key of current record 465

**NEXT RECORD** directive, end of SELECT 226

**RCD( )** function, return next record 508

**READ RECORD** directive, read from file 275

**REFILE** directive, clear data from file 278

**REMOVE** directive, delete from file 281

**RNO( )** function, next record number 514

**SELECT** directive, select/query records 299

system limits 825–826

**UPDATE** directive, update record in file 351

**WRITE** directive, add/update data in file 383

**WRITE RECORD** directive, write record 386

'RECTANGLE' mnemonic 635

**REDIM** directive, redimension array 277

'RED' & '\_RED' mnemonics 635

**REF( )** function, control reference count 512

**REFILE** directive, clear data from file 278

**RELEASE** directive, terminate ProvideX 279

**REM** directive, remark 280

remote processing

[RPC] Remote Process Control 797–798

**PROCESS SERVER** directive 258

See networks

**REMOVE** directive, delete record from file 281

**RENAME..INDEX** directive, rename keys 285

**RENAME CLASS** directive, OOP class 283

**RENAME CONTROL** directive, change CTL values 284

**RENAME** directive, change a file's name 282

**RENUMBER** directive, change line numbers 286

**REPEAT** directive, repetitive execution 287

reserved words 827

reset

**BEGIN** directive, reset files and variables 32

**CLEAR** directive, reset variables 54

**RESET** directive, reset program state 288

resource library 626

restart ProvideX 328

**RESTORE** directive, reset data pointer 289

retries, setting default 695

**RETRY** directive, re-execute instruction 290

**RET** system variable, last OS error code 571

**RETURN** directive, return from subroutine 291

reverse video 592

Rexon, *See* language compatibility

'RI' parameter 683

'RL' mnemonic 636

'RM' mnemonic 636

**RND( )** function, return random number 513

**RND** system variable, random numbers 571

**RNO( )** function, return next record number 514

'RN' parameter 684

rounding control 293

'NR' parameter 676

'RN' parameter 684

'RS' parameter 684

**ROUND** directive 293

[RPC] special command tag 797–798

'RP' mnemonic 636

'RP' parameter 684

'RR' parameter 684

'RS' mnemonic 636

'RS' parameter 684

'RT' mnemonic 637

**RUN** directive, transfer/execute program 294

## S

'+S' & '-S' mnemonics 637

'Sn' mnemonic 637

**SAVE CONTROL** directive, save image 297

**SAVE DATA** directive, save constants 297



- @** **SAVE** directive, write program to file 295, 688
- SAVE FILE** directive, save bitmap file 298
- 'SB'** mnemonic 638
- 'SB'** parameter 686
- B** scientific notation 133
- 'SC'** parameter 686
- C** screen capture, **SAVE CONTROL** directive 296
- D** scrolling
- 'SCROLL'** mnemonic 638
- grid scroll modes 147
- H\_SCROLLBAR** directive 153–155
- scroll region 602
- scroll wheel control 688
- V\_SCROLLBAR** directive 365–367
- 'SD'** parameter 686
- H** search
- rules, **PREFIX** directive 249
- rules, ProvideX defaults 251
- search and replace utility 25, 177, 672
- I** sectors
- highest available, **HSA()** function 450
- size, **SSZ()** function 521
- J** security 17, 199, 317, 438, 537, 798
- K** **SELECT** directive, select/query records 299
- semicolon (;) 25
- L** **'SE'** mnemonic 638
- SEP()** function, return field separator 515
- M** **SEP** system variable, field delimiter 572
- N** Serial file type 22
- O** **SERIAL** directive, create sequential file 302
- P** server 797, 799–801
- Oracle server 786
- printer defined on server 376
- PROCESS SERVER** directive 258
- Q** session 19
- terminate ProvideX session 39, 264, 279
- See ProvideX
- R** **SET\_FOCUS** directive, set input focus 304
- SET\_NBF** directive, set Keyed I/O buffers 305
- SET\_PARAM** directive, system parameters 306
- S** **SETCTL** directive, GOSUB on CTL event 307
- T** **SETDAY** directive, change local date 308
- U** **SETDEV** directive, set device type name 309
- V** **SETDEV IOL** directive, change IOList 310
- W** **SETDEV KEY** directive, change keys 311
- X** **SETDEV PROGRAM** directive, set I/O program 312
- Y** **SETDEV SEP=** directive, change file SEP 313
- Z** **SETDEV TSK()** directive, add to **TSK()** List 314
- SETDRIVE** directive, change default drive 315
- SETERR** directive, set error transfer 316
- SETESC** directive, set interrupt handler 317
- SETFID** directive, set **FID(0)** definition 320
- SETMOUSE** directive, control/set mouse 321
- SETTIME** directive, set local time 323
- SETTRACE** directive, enable program trace 324
- 'SF'** mnemonic 639
- 'SF'** parameter 686
- SGN()** function, return sign of value 516
- SHORT\_FORM** directive, use short names 325
- SHOW** directive, show control 326
- 'SHOW'** mnemonic 639
- SID** system variable, system ID code 572
- SIN()** function, sine 517
- 'SIZE'** mnemonic 639
- 'SK'** parameter 687
- slashes (/ or \) 26
- 'SL'** mnemonic 640
- 'SL'** parameter 687
- SN'** mnemonic 640
- sort
- Sort, Keyed file type 22
- SORT** directive, create file for sorting 327
- SRT()** function, sort string 519
- sound effect 589
- special command tags 21, 769–806
- [DB2]** 770–774
- [DDE]** 776
- [DLL]** 778, 781
- [MYSQL]** 783–785
- [OCI]** 786
- [ODB]** 788, 791, 795–796
- [RPC]** 797–798
- [TCP]** 799
- [WDX]** 801
- specialty files 21, 737–767
- 'SP'** mnemonic 640
- spooler
- See printers
- 'SP'** parameter 687
- SQL 468, 681, 698, 770, 783, 786, 791
- [DB2], [OCI], [ODB], [MYSQL]** 769
- SQR()** function, square root 518
- square brackets 25
- 'SR'** mnemonic 641
- 'SR'** parameter 687
- SRT()** function, sort string 519
- SSN** system variable, system software ID 572



**@**  
**A**  
**B**  
**C**  
**D**  
**E**  
**F**  
**G**  
**H**  
**I**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**

'SS' parameter 688

SSZ( ) function, return sector size 521

START directive, restart session 328

statements 19

conventions regarding syntax *xiv*

LIST directive, convert statements 176

logical statement references (labels) 816

LST( ) function, convert statements 477

separators (; semicolon) 25

statement reference 26

system limit 825

WHILE..WEND directives, repeat 375

*See also* directives, lines

STATIC directive, runtime OOP properties 329

STK( ) function, program call stack 522

STOP directive, halt program execution 330

*See also* terminate

STP( ) function, strip characters 523

STR( ) function, convert numeric to string 525

string files, *See* Serial file type

strings

binary of string, DEC( ) function 414

compile string, CPL( ) 407

contents of variable, VIS( ) function 549

convert to numeric, NUM( ) function 492

convert via table, TBL( ) function 532

evaluate variable, EVS( ) function 430

even parity of string, GEP( ) function 449

extract portion of string, MID( ) function 480

generate string, DIM( ) function 415

global string variable, GBL( ) function 447

hex value of string, HTA( ) function 454

JST( ) function 461

justify, JST( ) function 461

length, LEN( ) function 473

line labels 26

literal strings 25

lowercase string, LCS( ) function 472

numeric to string, STR( ) function 525

odd parity value, GAP( ) function 446

PAD( ) function 496

scan for mask, MSK( ) function 486

scan for occurrence, POS( ) function 502

sort, SRT( ) function 519

strip characters, STP( ) function 523

substitute text, SUB( ) function 527

text height, TXH( ) function 544

to different values, CVS( ) function 412

truncate, PAD( ) function 496

uppercase string, UCS( ) function 546

SUB( ) function, substitute text 527

subprograms

CALL directive, transfer to subprogram 40

ENTER directive, specify arguments 119

error report 676

EXIT directive, terminate and return 124

interrupt processing 317

PERFORM directive, CALL with variables 243

remote processing 797

XEQ( ) function, in-line execute 551

subroutines

GOSUB.. directive, execute subroutine 141

interrupt processing 317

ON ... GOSUB directive, conditional 230

RETURN directive 291

sults 660

'SV' parameter 688

'SWAP' mnemonic 641

switch

SWITCH directive, branch control 331

END SWITCH directive, end branching 115

LINE\_SWITCH directive, switch I/O 175

SWP( ) function, swap data 528

'SW' parameter 688

'SX' mnemonic 641

syntax

conventions *xiv*

punctuation 25

SYS( ) function, invoke OS command 163, 529

SYS system variable, operating system ID 573

system

functions 20, 389–554

limits 825

parameters 20, 306, 653–700

variables 20, 555–576

*See* ProvideX, operating system

SYSTEM\_HELP directive, Windows help 332

SYSTEM\_JRNL directive, file system

journalization 334

'SZ' parameter 688

**T**

'+T' & '-T' mnemonics 641

TABLE directive, define translation table 340

tables

accent conversion table 67, 70, 74

convert string via table, TBL( ) function 532

date table 67, 70, 74

define system tables 67, 70, 74

input conversion table 614

lowercase table 67, 70, 74



- @
  - output conversion table 624
  - prefix table entry 92, 110
  - uppercase table 67, 70, 74
- A
- B
- C
- D
- E
- F
- G
- H
- I
- J
- K
- L
- M
- N
- O
- P
- Q
- R
- S
- T
- U
- V
- W
- X
- Y
- Z
  - output conversion table 624
  - prefix table entry 92, 110
  - uppercase table 67, 70, 74
- tags, special command tags 769–806
- TAN()** function, tangent 531
- '**TA**' parameter 689
- tasks
  - entry from task list, **TSK()** function 543
  - TSK()** function 314
- TBL()** function, convert string via table 532
- '**TB**' parameter 689
- TCB()** function, return task information 534
- TCP/IP 435, 440, 442, 696, 798
  - [**TCP**] special command tag 799, 825
  - PROCESS SERVER** directive 258
  - See networks
- '**TC**' parameter 689
- terminals
  - DEFTTY** directive, define terminal 82
  - display editing 582
  - INPUT** directive, get input from terminal 160
  - OBTAIN** directive, get terminal input 227
  - PREINPUT** directive, prime input queue 254
- terminate
  - BYE** directive, terminate session 39
  - END** directive, halt program execution 113
  - POP** directive, premature exit from stack, 245
  - ProvideX 39, 264, 279
  - QUIT** directive, terminate session 264
  - RELEASE** directive, terminate session 279
  - STOP** directive, halt program execution 330
  - subprogram 124
  - WAIT** directive, temporarily halt 372
- text
  - accent character 67, 70, 74
  - cursor 582
  - return height, **TXH()** function 544
  - return width, **TXW()** function 545
  - substitute text, **SUB()** function 527
  - '**TEXT**' mnemonic 642
  - '**TEXTWDW**' mnemonic 643
  - See also strings
- text files, See Serial file type
- Thoroughbred, See language compatibility
- '**TH**' parameter 690
- tick or apostrophe operator (') 823
- time
  - device time-out 662
  - SETTIME** directive, set local 323
  - TIM** system variable, since midnight 573
  - TME** system variable, since midnight 574
  - TMR()** function, timer 541
  - TMS** system variable, in 60 seconds 574
- '**TL**' parameter 690
- '**TN**' parameter 690
- touchscreen 37
- tracing
  - ENDTRACE** directive, end trace output 118
  - SETTRACE** directive, enable tracing 324
- translation
  - TABLE** directive, define translation table 340
  - TBL()** function, translation table 532
  - TRANSLATE** directive, translate variable 341
- transmission
  - [**TCP**] Transmission Control Protocol 799
  - checksums 408
- tree view list boxes 192
  - object properties 707
  - state indicators 731
- TRISTATE\_BOX** directive, tristate box 344, 346–348
  - object properties 707
- '**TR**' mnemonic 643
- TRX()** function, Radix-40 to ASCII 542
- TSK()** function, entry from task list 314, 543
- TSM** system variable, current error status 575
- '**TT**' parameter 691
- '**TU**' parameter 691
- '**TW**' mnemonic 644
- TXH()** function, text height 544
- '**TX**' parameter 691
- TXW()** function, text width 545
- type-ahead mode 593
- U
  - '**+U**' & '**-U**' mnemonics 644
  - '**UC**' mnemonic 644
  - UCP()** function, uncompress data 547
  - UCS()** function, return uppercase string 546
  - UID** system variable, current user ID 575
  - '**UL**' parameter 692
  - '**UM**' parameter 692
  - underscoring 593
  - universal naming convention (UNC) 417, 421, 757, 761
  - UNIX 372, 417, 421, 567, 694, 769
    - changing user or group IDs 67, 70
    - delimiter 558
    - FacetTerm session 443



- @** UNLOCK directive, remove exclusive use 349
- A** UNTIL directive, end of REPEAT loop 350
- B** UNT system variable, lowest available channel 576
- C** UPDATE directive, update record in file 351
- D** UPK() function, unpack numeric data 548
- E** 'UP' mnemonic 644
- F** uppercase
- G** DEF UCS directive, define uppercase table 67, 70, 74
- H** UCS() function, uppercase string 546
- I** USER\_LEX directive, define alternate directives 352
- J** user-defined functions 68
- K** utilities
- L** \*CMD 177, 477
- M** \*LEXEDIT 353
- N** \*UCP 654
- O** \*UFAC 169
- P** \*UFAR 169, 687
- Q** \*WindX.utl 801, 804
- R** ProvideX 25, 604, 608
- S** search utility 25
- T** third party 61, 71, 228
- V**
- '+V' & '-V' mnemonics 645
- V\_SCROLLBAR directive, control vertical scrollbar 365–367
- object properties 708
- values, *See* numeric values, strings
- VARDROP\_BOX directive, control variable drop box 354–356, 358–359
- object properties 708
- variables 25
- CHG() function, if variable changed 402
- GBL() function, global string variable 447
- IOLIST directive, specify variable list 165
- LET directive, assign value to variable 173
- LOCAL directive, designate local data 197
- LONG\_FORM directive, long names only 201
- reset 54, 87
- SETDEV IOL directive, alter IOList 310
- SHORT\_FORM directive, use short names 325
- string variables 25
- system variables 555–576
- Variable Definition file 196, 297
- VIA directive, assign variable indirectly 368
- VIN() / VIS() functions, contents of 549
- VARLIST\_BOX directive, control variable list box 360–364
- object properties 708
- VBX 61
- 'VC' parameter 692
- VIA directive, assign variable indirectly 368
- video
- MULTI\_MEDIA directive, interface 223
- VIDEO\_PALETTE directive, video colours 370
- See* Graphical User Interface, terminals
- \*VIEWER\* 752
- VIN() function, numeric from variable 549
- VIS() function, string from variable 549
- Vista, *See* Windows
- Vista-style GUI 586
- VLR (variable length record) 22
- 'VM' parameter 692
- 'VP' parameter 692
- 'VR' parameter 693
- 'VT' mnemonic 645
- 'VW' parameter 693
- W**
- '+W' & '-W' mnemonics 645
- WAIT directive, temporarily halt execution 372
- WAIT FOR EVENT directive 373
- 'WA' mnemonic 646
- 'WB' parameter 693
- 'WC' mnemonic 646
- 'WD' mnemonic 646
- 'WD' parameter 694
- [WDX] special command tag 801
- WebServer *xiii*, 825
- WEND directive, end of WHILE Processing 374
- 'WF' parameter 694
- 'WG' mnemonic 646
- WHILE.. directive, repeat statements 375
- 'WHITE' & '\_WHITE' mnemonics 646
- WHO system variable current userID 576
- 'WH' parameter 694
- wildcard characters (\*) 252
- \*WINDEV\* 756
- Queues 763
- 'WINDOW' mnemonic 647
- Windows 694, 769
- API Frame Styles 579
- clipboard 55
- DIR() function, get current directory 417
- DLL() / DLX() functions 418



**@** **DSK()** function, get current disk drive [421](#)  
**MULTI\_MEDIA** directive [223](#)  
 Multimedia Control Interface (MCI) [223](#)  
 printer [376, 760, 805](#)  
 repainting window region [579](#)  
 resources (GDI) [21, 582](#)  
 spooler [586](#)  
**SYSTEM\_HELP** directive, invoke help [332](#)  
**VIDEO\_PALETTE** directive, video colours [370](#)  
 Vista-style GUI [213](#)  
 Windows-only parameters [692](#)  
 XP-style GUI [213, 586](#)  
 WinX [xiii, 801–807](#)  
**\*WINPRT\*** [760](#)  
     directing output to PDF [656, 750](#)  
     Queues [763](#)  
**WINPRT\_SETUP** directive, windows printer [376](#)  
**'WI'** parameter [694](#)  
**'WK'** parameter [695](#)  
**'WL'** parameter [695](#)  
**'WM'** mnemonic [648](#)  
 working directory [62](#)  
**'WP'** mnemonic [648](#)  
**'WRAP'** mnemonic [648](#)  
**WRITE** directive, add/update data in file [383](#)  
**WRITE RECORD** directive, write record [386](#)  
**'WR'** mnemonic [648](#)  
**'WS'** mnemonic [648](#)  
**'WT'** parameter [695](#)  
**'WX'** mnemonic [649](#)  
**'WZ'** parameter [696](#)

**X**  
**\*X'** mnemonic [649](#)  
**'XC'** parameter [696](#)  
**XEQ()** function, in-line subprogram [551](#)  
**XFA()** function, extended field attributes [552](#)  
**'XF'** parameter [696](#)  
**'XI'** parameter [696](#)  
 XML [764](#)  
     **\*XML** [764](#)  
**XOR()** function, exclusive OR comparison [554](#)  
 XP, *See* Windows  
**'XP'** mnemonic [650](#)  
**'XS'** parameter [697](#)  
**'XT'** parameter [697](#)  
**@X()** / **@Y()** functions, coordinates [390–391, 544–545, 579](#)

## Y

**'YELLOW'** & **'\_YELLOW'** mnemonics [650](#)

## Z

**'+Z'** & **'-Z'** mnemonics [585, 650](#)

ZLib compression [59, 167, 404, 538, 547, 696](#)

**'ZP'** parameter [697](#)

**'ZX'** mnemonic [645, 651](#)