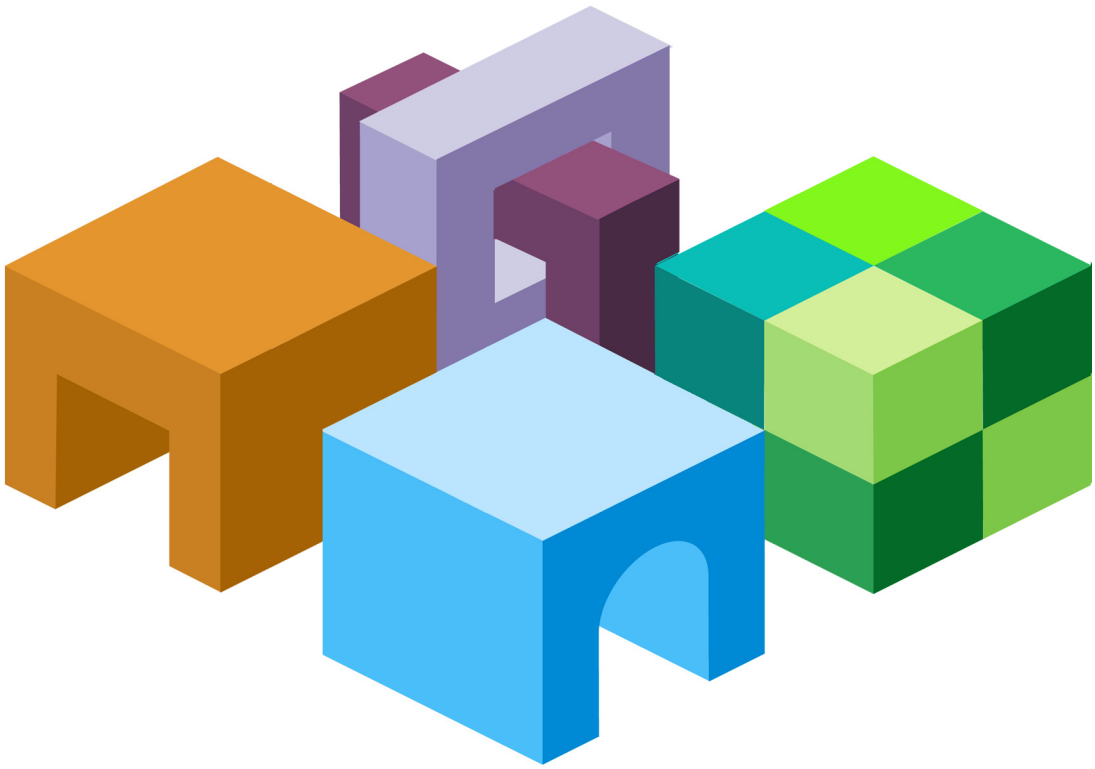

HYPERION® FINANCIAL MANAGEMENT – SYSTEM 9

RELEASE 9.3.1

LIBRARY OF FUNCTIONS

ORACLE | Hyperion

Financial Management LIBRARY OF FUNCTIONS, 9.3.1

Copyright © 2000, 2007, Oracle and/or its affiliates. All rights reserved.

Authors: Financial Management Information Development

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

1 Library of Functions Overview

The following custom functions are available as of Hyperion Financial Management Release 2.0. This library of functions contains management reporting functions as well as planning functions.

The scope of this library is limited to the available internal HS functions that are implemented for Hyperion Financial Management Release 2.0. These functions were available in Hyperion Financial Management 1.2 and have been updated for syntax changes in 2.0. We will continue to enhance this library to include additional custom functions and advanced business rules as necessary.

We have included two sample VB Script rules files for these functions: one for management reporting functions and one for planning functions. Users can copy and paste the relevant functions from the sample rules files to their own rules files.

For each custom function we have included a short description, the type of function, the return value, the syntax, a detailed description, an example, and a sample script.

If there is a need to modify the custom function, users should copy the custom function provided and rename the function before making the changes.

The following table provides a summary of the management reporting custom functions that are detailed in Chapter 2, "Management Reporting Functions".

Table 1 Management Reporting Functions

Custom Function	Description	Syntax	Function Type	Hyperion Enterprise Equivalent
Average	Calculates the financial average	Average (POV, Periods)	Function	AVE A12
Cumulative	Accumulates amounts from prior periods	Cumulative (POV, View, NumPeriod)	Function	CUM CTD YTD
Difference	Calculates the difference between current and opening	Difference (POV, View)	Function	DIF DFB
DSO	Calculates the days sales are outstanding	DSO (DSO, Debtor, Sales, DIP)	Procedure	
Opening	Carries opening balances forward	Opening (POV, View)	Function	OPE BASE BASEFLOW
Rate	Gets the relative exchange rate	Rate (ExchangeRate, Triangulation Currency)	Function	CrossRate

The following table provides a summary of the planning custom functions that are detailed in Chapter 3, "Business Rules Functions".

Table 2 Planning Functions

Custom Function	Description	Parameters	Function Type
Units_Rates	Units * rates (C=A*B)	Unit_Rates (Description, Units, Rates)	Procedure
Custom_Alloc	Allocates in the custom dimension	Custom_Alloc (Destination, Source, Factor, FactorN, FactorD, Elimination)	Procedure
Increase_Decrease	Increases or decreases the account by a percentage	Increase_Decrease (Destination, Source, Factor, Scale, Inverse)	Procedure
Pro_Rata_Ratio	The ratio between 2 accounts	Pro_Rata_Ratio (Destination, SourceN, SourceD)	Procedure
Spread	Spreads the total amount among all periods in the year	Spread (Destination, Source, Factor, FactorN, FactorD, Temp, Per)	Procedure

2 Management Reporting Functions

In This Chapter

The management reporting custom functions are described in this section. There are six management reporting custom functions included in Hyperion Financial Management Release 2.0.

Custom Functions

The custom functions are listed in alphabetical order. The following information is provided for each custom function:

- Both a short and detailed description
- A return value, if any
- A syntax
- An example using the function
- A sample script

Average

Calculates the average value for a specified fully defined account (Acc/C1/C2/C3/C4/ICP) across a number of periods.

Return Value

Returns a string of characters representing the correct expression to be used as part of the HS.EXP function.

Syntax

```
Average (PointOfView, Periods)
```

Table 3 Syntax of Average Function

Parameter	Valid Values
PointOfView	Valid combination of the RHS dimension which includes Account, Custom1....4, ICP members. For example, "A#CASH.C1#[None].I#[ICP Top]" For flow type accounts, the function will average only the periodic value.
Periods	It must be one of the three possible values: "YTD" - User specifies the year-to-date option to average the cumulative data from period one in the current year. "Periodic" - User specifies the periodic option to average the current and immediately prior period in the current year only. For the first period this value will be the same as the source. "[any whole positive number]" - User specifies a number of periods over which the average is to be calculated. For a rolling year average in a monthly category, the user would specify "12" here.

Detailed Description

This function calculates the average value of a given account over a specified number of prior periods. If the source is a balance type account, the average is based on the entered data. If the source is a flow type account, the average is based on the periodic data only.

The Average value will be derived differently based on the *Periods* parameter passed to the function.

- If the *Periods* parameter is "YTD", the average value will be the sum of all periods in the current year up to the current divided by the current period number.
- If the *Periods* parameter is "Periodic", the average value will be the sum of the current and immediately prior periods divided by 2. If the current period is the first period of the year, the average value will be the same value as the source.
- If the *Periods* parameter is a number, the average value will be the sum of the current and each preceding period for the specified number of periods, divided by the specified number.

Example

The account SALES will return the following values for Jan, Feb, and Mar, 2001 depending on the *Periods* parameter used in the Average custom function. The default view set for the scenario being processed is YTD.

Table 4 Example of Average Function

	Oct2000	Nov2000	Dec2000	Jan2001	Feb2001	Mar2001
A#Sales	9,000	10,500	11,700	800	1,900	3,200
Average(“A#Sales”, “YTD”)				800	950	1,067
Average(“A#Sales”, “Periodic”)				800	950	1,200
Average(“A#Sales, “3”)				1,167	1,033	1,067

Sample Script

```
' sample statement written in the calling routine
Sub Calculate()
Hs.Exp "A#AVG_SALES = " & Average("A#Sales", "12")
End Sub
' programming of the AVERAGE function
FUNCTION Average(strPOV,strPERIOD)
DIM nPERIOD
DIM strCUM
DIM i
strPOV = UCASE(strPOV)
strPERIOD = UCASE(strPERIOD)
IF strPERIOD = "PERIODIC" THEN
IF HS.PERIOD.ISFIRST = TRUE THEN
nPERIOD = 1
ELSE
    nPERIOD = 2
END IF
ELSEIF strPERIOD = "YTD" THEN
nPERIOD = HS.PERIOD.NUMBER()
ELSEIF CINT(strPERIOD) > 0 THEN
nPERIOD = CINT(strPERIOD)
ELSE
EXIT FUNCTION
END IF
FOR i = 0 TO nPERIOD-1
IF i = 0 THEN
strCUM = strPOV &".W#PERIODIC"
ELSE
strCUM = strCUM &"+"& strPOV &".W#PERIODIC.P#CUR-"&i
END IF
NEXT
Average = "(("& strCUM &")/"& nPERIOD &")"
END FUNCTION
```

Cumulative

Calculates the total of the preceding period's values for a specified account.

Return Value

Returns a string of characters representing the correct expression to be used as part of the HS.EXP function.

Syntax

`Cumulative (`*`PointOfView, View, NumPeriod`*`)`

Table 5 Syntax of Cumulative Function

Parameter	Valid Values
PointOfView	Valid combination of the RHS dimension which includes Account, Custom1....4, ICP members. For example, "A#CASH.C1#[None].I#[ICP Top]"
View	It must be one of the 3 possible values: " " (double quote) - Based on the default view defined for the scenario being processed (either YTD or Periodic). "YTD" - User specifies the Year-to-date option, which overrides the default view set for the scenario. "Periodic" - User specifies the periodic option, which overrides the default view set for the scenario.
NumPeriod	A whole number representing the number of periods in the current scenario to accumulate, starting with the current period. If NumPeriod is 0 or negative, the function will aggregate from the beginning of the current year.

Detailed Description

This function calculates the sum of either the periods specified or or calculates the sum year to date for the specified account. By default, the view of the data accumulated will be the scenario default; however, the user may wish to override this for flow type accounts.

- If the *View* parameter is "YTD", the function will accumulate the year-to-date values.
- If the *View* parameter is "Periodic", the function will accumulate the periodic values.
- If the *View* parameter is blank (" "), the function will accumulate the data using the scenario default view.

Example

The account CASH will return the following values for Jan, Feb, and Mar, 2001 depending on the *Number* parameter used in the Cumulative function.

The account SALES will return the following values for Jan, Feb, and Mar, 2001 depending on both the *View* and *Number* parameters used in the Cumulative function. The default view set for the scenario being processed is YTD.

Table 6 Example of Cumulative Function

	Oct2000	Nov2000	Dec2000	Jan2001	Feb2001	Mar2001
A#Cash	1,000	1,500	1,200	800	1,100	1,300
Cumulative(“A#Cash”, “”, 0)				800	1,900	3,200
Cumulative(“A#Cash”, “”,3)				3,500	3,100	3,200
A#Sales	9,000	10,500	11,700	800	1,900	3,200
Cumulative(“A#Sales”, “”,0)				800	2,700	5,900

	Oct2000	Nov2000	Dec2000	Jan2001	Feb2001	Mar2001
Cumulative("A#Sales", "Periodic",0)				800	1,900	3,200
Cumulative("A#Sales", "Periodic",3)				3,500	3,100	3,200

Sample Script

```
' sample statement written in the calling routine
Sub Calculate()
HS.EXP "A#TOT_Cash ="&Cumulative("A#Cash"," ",0)
End Sub
' programming of the Cumulative function
Function Cumulative(StrPov, StrVIEW, nPERIOD)
DIM strCUM
DIM i
IF nPERIOD <= 0 THEN
nPERIOD = HS.PERIOD.NUMBER() - 1
ELSE
nPERIOD = nPERIOD - 1
END IF
IF strVIEW = "" THEN
strVIEW = HS.SCENARIO.DEFAULTVIEW("")
END IF
strPOV = UCASE(strPOV)
strVIEW = UCASE(strVIEW)
IF strVIEW = "PERIODIC" THEN
strVIEW = ".W#PERIODIC"
ELSEIF strVIEW = "YTD" THEN
strVIEW = ".W#YTD"
ELSE
EXIT FUNCTION
END IF
FOR i = 0 TO nPERIOD
IF i = 0 THEN
strCUM = strPOV & strVIEW
ELSE
strCUM = strCUM &"+"& strPOV & strVIEW &".P#CUR-"&i
END IF
NEXT
  Cumulative = "("& strCUM &")"
END FUNCTION
```

Difference

Calculates the difference between the current period value and the opening value.

Return Value

Returns a string of characters representing the correct expression to be used as part of the HS.EXP function.

Syntax

`Difference (PointOfView, View)`

Table 7 Syntax of Difference Function

Parameter	Valid Values
PointOfView	Valid combination of the RHS dimension which includes Account, Custom1....4, ICP members. For example, "A#CASH.C1#[None].I#[ICP Top]"
View	It must be one of the 3 possible values: " " (double quote) - Based on the default view defined for the scenario being processed (either YTD or Periodic). "YTD" - User specifies the Year-to-date option, which overrides the default view set for the scenario. "Periodic" - User specifies the periodic option, which overrides the default view set for the scenario.

Detailed Description

This function calculates the difference between the value of the current period and the opening value. (Current - Opening)

The opening value will be derived differently based on the *View* parameter passed to the function.

- If the *View* parameter is "YTD", the opening value will be retrieved from the last period of the prior year.
- If the *View* parameter is "Periodic", the opening value will be retrieved from the prior period of the current year. If the current period is the first period of the year, the opening value will be retrieved from the last period of the prior year.
- If the *View* parameter is blank (" "), the opening value will be based upon the default data view of the scenario.

Example

The account CASH will return the following values for Jan, Feb, and Mar, 2001 depending on the *View* parameter used in the Difference function. The default view set for the scenario being processed is YTD. The Difference function subtracts the opening value from the current period value.

Table 8 Example of Difference Function

	Dec2000	Jan2001	Feb2001	Mar2001
A#Cash	900	1,200	1,100	1,500
Difference("A#Cash", "")		300	200	600
Difference("A#Cash", "YTD")		300	200	600
Difference("A#Cash", "Periodic")		300	-100	400

Sample Script

```
' sample statement written in the calling routine
Sub Calculate()
Hs.Exp "A#DiffCash = " & Difference("A#Cash", "YTD")
End Sub
' programming of the DIFFERENCE function
FUNCTION DIFFERENCE(strPOV,strVIEW)
IF strVIEW = "" THEN
strVIEW = HS.SCENARIO.DEFAULTVIEW (“”)
END IF
strPOV = UCASE(strPOV)
strVIEW = UCASE(strVIEW)
IF strVIEW = "PERIODIC" THEN
DIFFERENCE = "("&strPOV &"-"& strPOV & ".P#PRIOR" &")"
ELSEIF strVIEW = "YTD" THEN
DIFFERENCE = "("&strPOV &"-"& strPOV & ".Y#PRIOR.P#LAST" &")"
ELSE
EXIT FUNCTION
END IF
END FUNCTION
```

DSO - Days Sales Outstanding

Calculates the number of days sales in the current period debtors using the exhaustion method.

Return Value

This routine calculates a single value representing the amount of days sales contained within the current period trade debtors figure. The DSO sub-routine included here makes certain assumptions:

- Both Debtors and Sales are positive figures.
- The parameters supplied are fully defined points of view (for example, Account/C1/C2/C3/C4/ICP) because the routine uses the HS.GETCELL function.
- The routine will calculate the days going back as far as possible in time. However, it will stop if the periodic sales value for any period is a negative or zero value.

Syntax

`CALL DSO (strDSO,strDEBTOR,strSALES,strDIP)`

Table 9 Syntax of DSO Function

Parameter	Valid Values
strDSO	Fully defined account with custom and intercompany dimensions. This account is the destination for the calculation.
strDEBTOR	Fully defined account with custom and intercompany dimensions. This account is the source for the current period trade debtors.

Parameter	Valid Values
strSALES	Fully defined account with custom and intercompany dimensions. This account is the source for the sales. Specifically exclude references to frequency.
strDIP	Fully defined account with custom and intercompany dimensions. This account is the source for the number of days in the period. This is assumed to be in the [None] entity.

Detailed Description

The routine takes the values in the debtors account (parameter 2) and sales account (parameter 3) for the current period and compares them. If either are zero or negative, the calculation stops. For each successive period where the debtors value exceeds that of the cumulative sales (working backwards from the current period), the routine will add the number of days for that period as specified in the days in the period account (parameter 4) to a running total.

When all the debtors value has been "exhausted" in this way, the final period's days are calculated as a proportion of the unexpired debtors against the periodic sales value.

Finally, the routine posts the running total to the destination account (parameter 1).

Example

The example calculates the total days outstanding for the months shown.

Table 10 Example of DSO Function

Month	Debtors	Period Sales	Days in Month	Formula for DSO	Total DSO
September	12,000	2,500	30	100%	30
August		1,750	31	100%	31
July		2,250	31	100%	31
June		2,500	30	100%	30
May		2,000	31	100%	31
April		2,250	30	2000/2250	26.7
Total					179.7

Sample Script

```
' Use within the calculation section:
' 1. Standard use
CALL DSO("A#DSO","A#TradeDebtors.C1#AllAges.C2#[None].I#[ICP
Top]","A#TotalSales.C1#[None].C2#AllProducts.I#[ICP Top]","A#DIP")
' 2. Use with a common custom dimension
```

```
set vPRODUCT = ARRAY("C2#PRODUCT1","C2#PRODUCT2", ….. ,"C2#PRODUCTn")
FOR EACH iITEM IN vPRODUCT
CALL DSO("A#DSO."&iITEM,"A#TradeDebtors.C1#AllAges.I#[ICP
Top]."&iITEM,"A#TotalSales.C1#[None].I#[ICP Top]."&iITEM,"A#DIP")
NEXT
' Actual script of Sub-routine
SUB DSO(strDSO,strDEBTOR,strSALES,strDIP)
DIM vTEST
DIM vDSO
DIM vCOUNT
DIM vXS_1
DIM vXS
HS.CLEAR(strDSO)
vTEST = HS.GETCELL(strDEBTOR) * HS.GETCELL(strSALES&".W#Periodic") *
HS.GETCELL(strDIP&".E#[None]")
' checks if any of the parameters are zero (uses principle of X * 0 = 0)
IF vTEST = 0 THEN
EXIT SUB
ELSE
vDSO = 0
vCOUNT = 0
vXS_1 = HS.GETCELL(strDEBTOR)
vXS = vXS_1 - HS.GETCELL(strSALES&".W#Periodic")
' ensures that periodic sales are not negative or zero
WHILE vXS > 0 AND vXS_1 > vXS
vDSO = vDSO + HS.GETCELL(strDIP&".E#[None].P#CUR-" &vCOUNT)
vCOUNT = vCOUNT + 1
vXS_1 = vXS
vXS = vXS - HS.GETCELL(strSALES&".W#Periodic.P#CUR-" &vCOUNT)
WEND
IF vXS = vXS_1 THEN
vCOUNT = vCOUNT - 1
END IF
vDSO = vDSO + (vXS_1 / HS.GETCELL(strSALES&".W#Periodic.P#CUR-" &vCOUNT)
*HS.GETCELL(strDIP&".E#[None].P#CUR-" &vCOUNT))
IF vDSO < 0 THEN
vDSO = 0
END IF
END IF
HS.EXP strDSO &"="& vDSO
END SUB
```

Opening

Retrieves the opening value for a specified, fully defined account (Acc/C1/C2/C3/C4/ICP).

Return Value

This function returns a string of characters representing the correct expression to be used as part of the HS.EXP function.

Syntax

```
Opening (PointOfView, View)
```

Table 11 Syntax of Opening Function

Parameter	Valid Values
PointOfView	Valid combination of the RHS dimension which includes Account, Custom1....4, ICP members. For example, "A#CLOSE.C1#[None].I#[ICP Top]"
View	It must be one of the 3 possible values: " " (double quote) - Based on the default view defined for the scenario being processed (either YTD or Periodic). "YTD" - User specifies the Year-to-date option, which overrides the default view set for the scenario. "Periodic" - User specifies the Periodic option, which overrides the default view set for the scenario.

Detailed Description

This function calculates the opening value of a given account. The opening value will be derived differently based on the *View* parameter passed to the function.

- If the *View* parameter is "YTD", the opening value will be retrieved from the last period of the prior year.
- If the *View* parameter is "Periodic", the opening value will be retrieved from the prior period of the current year. If the current period is the first period of the year, the opening value will be retrieved from the last period of the prior year.
- If the *View* parameter is blank (" "), the opening value will be based upon the default data view of the scenario.

Example

The account FA_COST will return the following values for Jan, Feb, and Mar, 2001 depending on the *View* parameters used in the Opening function. The default view set for the scenario being processed is YTD.

Table 12 Example of Opening Function

	Dec2000	Jan2001	Feb2001	Mar2001
A#FA_COST	900	1,200	1,100	1,500
Opening("A#FA_COST", " ")		900	900	900
Opening("A#FA_COST", " YTD")		900	900	900
Opening("A#FA_COST", "Periodic ")		900	1,200	1,100

Sample Script

```
' sample statement written in the calling routine
Sub Calculate()
Hs.Exp "A#Open_FA_Cost = " & Opening("A#FA_Cost", "YTD")
End Sub
' programming of the OPENING function
FUNCTION OPENING(strPOV,strVIEW)
IF strVIEW = "" THEN
strVIEW = HS.SCENARIO.DEFAULTVIEW (“”)
END IF
strPOV = UCASE(strPOV)
strVIEW = UCASE(strVIEW)
IF strVIEW = "PERIODIC" THEN
OPENING = strPOV &".P#PRIOR"
ELSEIF strVIEW = "YTD" THEN
OPENING = strPOV &".Y#PRIOR.P#LAST"
ELSE
EXIT FUNCTION
END IF
END FUNCTION
```

Rate

Calculates the relative exchange rate between a parent and child and returns the value as a multiplier.

Return Value

This function returns a value to be used as part of an HS.EXP function, usually in the translation section.

Syntax

`Rate (`*`ExchangeRate, TriangulationCurrency`*`)`

Table 13 Syntax of Rate Function

Parameter	Valid Values
ExchangeRate	A main account of the type "CurrencyRate" specified as an account string, without reference to custom or intercompany dimensions. For example, "A#EOP_RATE"
TriangulationCurrency	This is either a valid currency label as a string or double quotes (" "). When specifying a currency, it is not necessary to reference any custom dimension.

Detailed Description

- This function calculates the relative exchange rate between a parent and child, returning a value as a multiplier. The value will be calculated based on the *TriangulationCurrency* parameter passed to the function.
- If the *TriangulationCurrency* parameter is a valid currency label, the cross rate will be based on this currency.
- If the *TriangulationCurrency* parameter is blank (" "), the function first searches for a valid direct rate, and if none is found will then use Triangulation against the application currency.
- If no rate values can be found, the function will return 1.

The following tables show the methods of searching for the data and the order in which the search is made. The order is represented by a number in parentheses, for example (1). In each case, the search is made first in the child entity and, if no data is found, then from the "[None]" entity.

In the following table, either the currency of the child or of the parent is the same as the Triangulation currency, or if Triangulation is blank, the application currency.

Table 14 Rate Example — Triangulation Currency Same

		Custom 1 dimension rates	
		Child	Parent
Custom 2 dimension rates	Child		(2)
Parent	(1)		

In the following table, Triangulation has been specified and is not the same as either the child or parent currencies.

Table 15 Rate Example — Triangulation Currency Different

		Custom 1 dimension rates		
		Child	Parent	Triangulation
Custom 2 dimension rates	Child			(2)
Parent				
Triangulation		(1)		

In the following table, Triangulation has not been specified and the application currency is different from both the child and parent currencies.

Table 16 Rate Example — Triangulation Not Specified

	Custom 1 dimension rates		
	Child	Parent	Application

		Custom 1 dimension rates		
Custom 2 dimension rates	Child		(2)	(4)
Parent	(1)			
Application		(3)		

Example

The application currency is Euros, and we are translating a French child to a US parent using the following rates entered in the [None] entity against the C2#EURO:

Table 17 Example of Rate Function

	Opening Rate	Closing Rate
C1#FFR	0.16000	0.16500
C1#USD	1.15862	1.15785

The following function multiplies the opening balance account by the difference between the relative ending and opening rates. This is useful when calculating movement analyses if the translation is not consistently between the local and application currencies.

```
HS.EXP "A#FXO = A#OPEN * (" & RATE("A#EOP_RATE"," ") & "-" & RATE
("A#OPE_RATE"," ") &")"
```

For the previous example, if the value in the OPEN account for the child is FFR 10,000,000, the value in the US parent FXO account will be USD 44,102 [10,000,000 * (0.165 /1.15785 - 0.16 / 1.15862)].

Sample Script

```
' sample statement written in the calling routine
SUB TRANSLATE()
HS.TRANS "A#FXO","A#FXO","A#EOP_RATE",“”
HS.EXP "A#FXO = A#OPEN * (" & RATE("A#EOP_RATE"," ") & "-" & RATE
("A#OPE_RATE"," ") &")"
END SUB
' programming of the RATE function
FUNCTION RATE(sRATE,sTRI)
DIM sCCUR, sPCUR, sACUR, bRET, retValue, s3rdCUR
DIM i
sRATE = UCASE(sRATE)
sTRI = UCASE(sTRI)
sCCUR = UCASE(HS.ENTITY.DEFCURRENCY(“”))
sPCUR = UCASE(HS.VALUE.CURRENCY)
sACUR = UCASE(HS.APPSETTINGS.CURRENCY)
retValue = 0
' check whether there is a triangulation specified, or if triangulation or
application currencies are the same as either parent or child and set up
the select case
```

```
IF sTRI = sCCUR OR sTRI = sPCUR OR (sTRI = " " AND (sACUR = sCCUR OR sACUR
= sPCUR)) THEN
i = 1
ELSEIF sTRI <> " " THEN
i = 2
ELSE
i = 3
END IF
SELECT CASE i
CASE 1
' bRET is a boolean that returns true if data is found. First search the
child...
' ...then search the [None] entity
bRET = GETVALUECP(".V#<Entity Currency>",retValue,sRATE,sCCUR,sPCUR)
IF NOT bRET THEN
bRET = GETVALUECP(".E#[None]",retValue,sRATE,sCCUR,sPCUR)
END IF
CASE 2
' use a dynamic parameter name for ease of writing the triangulation checks
s3rdCUR = sTRI
bRET = GETVALUE3(".V#<Entity Currency>",retValue,sRATE,sCCUR,sPCUR,s3rdCUR)
IF NOT bRET THEN
bRET = GETVALUE3(".E#[None]",retValue,sRATE,
sCCUR,sPCUR,s3rdCUR)
END IF
CASE 3
' this case is used when the 2nd parameter is blank and is the most
complex.
' first check direct rates in the child…
' … then check triangulation against application currency in the child
' then check direct rates in [None].
'… finally check triangulation in [None]
s3rdCUR = sACUR
bRET = GETVALUECP(".V#<Entity Currency>",retValue,sRATE,sCCUR,sPCUR)
IF NOT bRET THEN
bRET = GETVALUE3(".V#<Entity Currency>",retValue,sRATE,sCCUR,sPCUR,s3rdCUR)
IF NOT bRET THEN
bRET = GETVALUECP(".E#[None]",retValue,sRATE,sCCUR,sPCUR)
IF NOT bRET THEN
bRET = GETVALUE3(".E#[None]",retValue,
sRATE,sCCUR,sPCUR,s3rdCUR)
END IF
END IF
END IF
END SELECT
IF bRET THEN
RATE = retValue
ELSE
RATE = 1
END IF
END FUNCTION
FUNCTION GETVALUECP(sENTITY,sVALUE,sRATE,sCCUR,sPCUR)
' this sub-function is used when comparing direct rates between child and
parent
GETVALUECP = FALSE
' check if data exists for direct rate child to parent. If it does return
it.
```

```
' if no direct child to parent rate check for indirect parent to child
rate...
' return the inverse of the indirect rate.
IF HS.GETCELL(sRATE & ".C1#" & sCCUR & ".C2#" & sPCUR & sENTITY) <> 0 THEN
sVALUE = CDBL(HS.GETCELL(sRATE & ".C1#" & sCCUR & ".C2#" & sPCUR &
sENTITY))
GETVALUECP = TRUE
ELSEIF HS.GETCELL(sRATE & ".C1#" & sPCUR & ".C2#" & sCCUR & sENTITY) <> 0
THEN
sVALUE = CDBL(1 / HS.GETCELL(sRATE & ".C1#" & sPCUR & ".C2#" & sCCUR &
sENTITY))
GETVALUECP = TRUE
END IF
END FUNCTION
FUNCTION GETVALUE3(sENTITY,sVALUE,sRATE,sCCUR,sPCUR,s3rdCUR)
' this sub-function is used when triangulating
' check if data exists for direct rate child to triangulation…
' … if it does return the direct relative rate child to parent…
' if no direct child to triangulation rate check for indirect triangulation
to child rate…
' … return the inverse of the indirect relative rates.
GETVALUE3 = FALSE
IF HS.GETCELL(sRATE & ".C1#" & sCCUR & ".C2#" & s3rdCUR & sENTITY) <> 0
THEN
sVALUE = CDBL(HS.GETCELL(sRATE & ".C1#" & sCCUR & ".C2#" & s3rdCUR &
sENTITY) / HS.GETCELL(sRATE & ".C1#" & sPCUR & ".C2#" & s3rdCUR & sENTITY))
GETVALUE3 = TRUE
ELSEIF HS.GETCELL(sRATE & ".C1#" & s3rdCUR & ".C2#" & sCCUR & sENTITY) <> 0
THEN
sVALUE = CDBL(HS.GETCELL(sRATE & ".C1#" & s3rdCUR & ".C2#" & sPCUR &
sENTITY) / HS.GETCELL(sRATE & ".C1#" & s3rdCUR & ".C2#" & sCCUR & sENTITY))
GETVALUE3 = TRUE
END IF
END FUNCTION
```

3 Business Rules Functions

In This Chapter

The business rules custom functions are described in this section. There are five business rules custom functions included in Hyperion Financial Management Release 2.0.

Custom Functions

The custom functions are listed in alphabetical order. The following information is provided for each custom function:

- Both a short and detailed description
- A return value, if any
- A syntax
- An example using the function
- A sample script

Custom_Alloc

This function allocates a Source point of view (POV) to a Destination POV using a Factor POV as the basis of Allocation, with the option to reverse post the total allocated amount to an Elimination POV. This function is designed for custom dimension allocations.

Return Value

No return value.

Syntax

`Custom_Alloc(`*`Destination`*`,`*`Source`*`,`*`Factor`*`,`*`FactorN`*`,`*`FactorD`*`,`*`Elimination`*`)`

Table 18 Syntax of Custom_Alloc Function

Parameter	Valid Values
Destination	A valid destination POV. That is, a valid combination of Account, ICP and Custom 1-4 members.
Source	A valid source POV. That is, a valid combination of dimension members. *Source* is the amount that is to be allocated.
Factor	A valid source POV. *Factor* is the Account used to store the allocation factor.
FactorN	A valid source POV. *FactorN* is the numerator factor used as the basis for allocation.
FactorD	A valid source POV. *FactorD* is the denominator factor used as the basis for allocation.
Elimination	A valid source POV. *Elimination* may be an empty string (""), in which case this parameter is ignored. If the *Elimination* parameter is set, the amount posted to the *Destination POV* will be multiplied by -1 and posted to the Elimination POV.

Detailed Description

This function allocates a Source POV to a Destination POV using a Factor POV as the basis of allocation, with the option to reverse post the total allocated amount to an Elimination POV. This function is designed for custom dimension allocations.

The *Factor* parameter stores the result of *FactorN* divided by *FactorD*. This is required to enable the factor to refer to entities other than the current entity.

If the entity in the Source POV is a parent, that parent must be consolidated before executing the calculation at the child level. If the parent currency is different from the child currency, then a translation of all relevant currencies must also be run before executing the calculation at the child level.

It is recommended that variables are set in the calling routine and passed to the Custom_Alloc function, which define the Destination, Source, Factor, FactorN, FactorD and Elimination POVs. It is also recommended that the variable names in the calling routine be set to be the same as the Custom_Alloc function.

The *Elimination* parameter may be an empty string (""), in which case this parameter is ignored. If the *Elimination* parameter is set, the amount posted to the Destination POV will be multiplied by -1 and posted to the Elimination POV.

Example

The account Telephone is allocated to Products based on a ratio of Products Sales to Total Sales. The inverse of the allocated amount will be posted to account Allocations.

Table 19 Example of Custom_Alloc Function

	Jan2001	**Feb2001**	**Mar2001**
A#Telephone.C1#[None]	100	300	400

	Jan2001	Feb2001	Mar2001
A#Sales".C1#Product1	1000	1000	1000
A#Sales.C1#Product2	1000	2000	3000
A#Sales.C1#TotalProducts	2000	3000	4000
Custom_Alloc("A#Telephone","A#Telephone.C1#[None]", "A#Factor", A#Sales", "A#Sales.C1#TotalProducts", "A#ProductAllocations.C1#[None]")			
A#Factor.C1#Product1	0.50	0.33	0.25
A#Factor.C1#Product2	0.50	0.66	0.75
A#Telephone.C1#Product1	50	100	100
A#Telephone.C1#Product2	50	200	300
A#ProductAllocations.C1#[None]	-100	-300	-400

The result returned from the CUSTOM_ALLOC function is as follows:

```
HS.EXP "A#Factor = A#Sales / A#Sales.C1#TotalProducts"
HS.EXP "A#Telephone = A#Telephone.C1#[None] * A#Factor"
HS.EXP "A#Allocations.C1#[None] = (A#Telephone.C1#[None] * -1)"
```

Sample Script

This script contains the following information:

- A sample statement written in the calling routine.
- Variables set in the calling routine and passed to the Custom_Alloc function.
- Variable names in the calling routine set to be the same as the Custom_Alloc function.

```
Sub Calculate()
Dim Destination
Dim Source
Dim Elimination
Dim Factor
Dim FactorN
Dim FactorD
Dim C1list
Dim C1item
C1list = HS.Custom1.List("Alloc")
For Each C1item in C1list
Source = "A#Telephone.C1#[None]"
Destination = "A#Telephone.C1#" & C1item
Factor = "A#Factor.C1#" & C1item
FactorN = "A#Sales.C1#" & C1item
FactorD = "A#Sales.C1#TotalProducts"
Elimination = "A#ProductAllocations.C1#" & C1item
```

```
Call Custom_Alloc(Destination,Source,Factor,FactorN,
FactorD,Elimination)
Next
End Sub
' Beginning of the Custom_Alloc function
Sub Custom_Alloc(Destination,Source,FactorN,FactorD,
Elimination)
HS.Clear Factor
HS.Exp Factor & " = " & FactorN & "/" & FactorD
HS.EXP Destination & " = " & Source & " * " & Factor
If Elimination <> "" Then
HS.EXP Elimination & " = " & Source & " * -1 * " & Factor
End If
End Sub
```

Increase_Decrease

This function increases or decreases a Destination POV by a percentage Factor. The percentage factor may be taken from either a Source POV, a VBScript constant or a VBScript variable.

Return Value

No return value.

Syntax

`Increase_Decrease(`*`Destination,Source,Factor,Scale,Inverse`*`)`

Table 20 Syntax of Increase_Decrease Function

Parameter	Valid Values
Destination	A valid destination POV. That is, a valid combination of Account, ICP and Custom 1-4 members.
Source	A valid source POV. That is, a valid combination of dimension members. *Source* is the amount that is to be allocated.
Factor	A valid source POV, constant, or variable.
Scale	Integer value 1 or 100. Factor is divided by scale.
Inverse	True or False. True reverses the sign of the Factor. This can be used to generate a decrease where the Factor is stored as a positive number (or Visa Versa). False takes the stored sign of the Factor to determine an increase or decrease.

Detailed Description

This function increases or decreases a Destination POV by a percentage factor. The percentage factor may be taken from a Source POV, a VBScript constant or a VBScript variable.

In general, the Source POV will be the same as the Destination POV. However, the Source POV may also be different from the Destination POV.

The *Scale* parameter is used to scale down the factor, if required. This will be relevant where the factor is taken from a Source POV and the factor is stored in a non-scaled form (for example, 50% is stored as 50 and not 0.50).

The *Inverse* parameter is used to reverse the sign of the factor. This will be relevant where the factor is taken from a Source POV and the factor is stored as an absolute number. If the *Inverse* parameter is set to True, the factor will be multiplied by -1. If the *Inverse* parameter is set to False, the factor will not be multiplied -1.

Example

In this example, the account Telephone is increased by 10%.

Table 21 Example of Increase_Decrease Function

	Jan2001	Feb2001	Mar2001
A#Telephone	100	300	400
A#Factor/C1[None]	10	10	10
Increase_Decrease("A#Telephone", "A#Telephone", "A#Factor.C1#[None]", 100,False)			
A#Telephone	110	330	440

The result returned from the INCREASE_DECREASE function is as follows:

```
HS.EXP "A#Telephone = A#Telephone * (1+ (A#Factor.C1#[None]/100))"
```

Sample Script

- A sample statement written in the calling routine.
- Variables set in the calling routine and passed to the Increase_Decrease function.
- Variable names in the calling routine set to be the same as the Increase_Decrease function.

```
Sub Calculate()
Dim Destination
Dim Source
Dim Factor
Dim Scale
Dim Inverse
Destination = "A#Telephone"
Source = "A#Telephone"
Factor = "A#Factor.C1#[None]"
Scale = "100"
Inverse = False
Call Increase_Decrease(Destination,Source,Factor,Scale,
Inverse)
End Sub
```

```
' Beginning of the Increase_Decrease function
Sub Increase_Decrease(Destination,Source,Factor,Scale,Inverse)
If Inverse = False Then
HS.EXP Destination & " = " & Source & " *
(1 + (" & Factor & " / " & Scale & "))"
Else
HS.EXP Destination & " = " & Source & " *
(1 + ((" & Factor & " * -1) / " & Scale & ))"
End If
End Sub
```

Pro_Rata_Ratio

This function calculates the ratio between two source POVs (C = A / B).

Return Value

No return value.

Syntax

`Pro_Rata_Ratio(`*`Destination`*`,`*`SourceN`*`,`*`SourceD`*`)`

Table 22 Syntax of Pro_Rata_Ratio Function

Parameter	Valid Values
Destination	A valid destination POV. That is, a valid combination of Account, ICP and Custom 1-4 members.
SourceN	A valid source POV. That is, a valid combination of dimension members. *SourceN* is the numerator of the ratio calculation.
SourceD	A valid source POV. *SourceD* is the denominator of the ratio calculation.

Detailed Description

This function calculates the ratio between two source POVs (C = A / B).

It is recommended that variables are set in the calling routine and passed to the Pro_Rata_Ratio function, which define the Destination, SourceN and SourceD POVs. It is also recommended that the variable names in the calling routine be set to be the same as the Pro_Rata_Ratio function. These recommendations are a suggested best practice approach.

It should be noted that HFM does not naturally calculate weighted average ratios for parent members. Parent member values will appear as an aggregation of child values. This will always result in a mathematically incorrect value for parent members. As such, it is recommended that aggregation be turned off for Ratio accounts.

Example

The account MarginPct will return the value of GrossMargin/TotalRevenues.

Table 23 Example of Pro_Rata_Ratio Function

	Jan2001	Feb2001	Mar2001
A#GrossMargin	1000	100	750
A#TotalRevenues	2000	400	1000
Pro_Rata_Ratio("A#GrossMargin","#TotalRevenues")	0.50	0.25	0.75

The result returned from the PRO_RATA_RATIO function is as follows:

```
HS.EXP "A#MarginPct = A#GrossMargin / A# TotalRevenues"
```

Sample Script

The script contains the following information:

- A sample statement written in the calling routine.
- Variables set in the calling routine and passed to the Pro_Rata_Ratio function.
- Variable names in the calling routine set to be the same as the Pro_Rata_Ratio function.

```
Sub Calculate()
Dim Destination  'Destination POV
Dim SourceN      'Source Numerator POV
Dim SourceD      'Source Denominator POV
Destination = "A#MarginPct"
SourceN = "A#GrossMargin"
SourceD = "A#TotalRevenues    "
Call Pro_Rata_Ratio(Destination,SourceN,SourceD)
End Sub
' Beginning of the Pro_Rata_Ratio function
Sub Pro_Rata_Ratio(Destination,SourceN,SourceD)
HS.EXP Destination & " = " & SourceN & " / " & SourceD
End Sub
```

Spread

This function allocates a single time period value (e.g. P#[Year]) of a Source Account to all periods of a Destination Account based on a profile defined in a Profile Account (e.g., Revenue profile, 4-4-5, etc.).

Return Value

No return value.

Syntax

`Spread(`*`Destination,Source,Factor,FactorN,FactorD,Temp,Per`*`)`

Table 24 Syntax of Spread Function

Parameter	Valid Values
Destination	A valid destination POV. That is, a valid combination of Account, ICP and Custom 1-4 members.
Source	A valid source POV. That is, a valid combination of dimension members. The Source POV must include a single time period, for example, P#[Year]. The single time period amount is the amount to be spread.
Factor	A valid source POV. *Factor* is the account used to store the allocation factor.
FactorN	A valid source POV. *FactorN* is the numerator factor used as the basis for spread allocation.
FactorD	A valid source POV. *FactorD* is the denominator factor used as the basis for spread allocation.
Temp	A valid destination Account. *Temp* is the account that temporarily stores the Source value.
Per	A period string that defines the name of the first period in the timeframe, for example, "January". The *Temp* value is stored in the first period and the parameter is required to refer to this in the calculation.

Detailed Description

This function allocates a single time period value (e.g. P#[Year]) of a Source POV to all periods of a Destination POV based on a profile defined in a Profile POV (for example, Revenue profile, 4-4-5, and so on).

Time-based allocations are particularly suited to budgeting applications where amounts are first entered for the total year, and then later allocated across time periods based on a suitable profile.

The Source POV must contain a single time period. The time period will generally be P#[Year], but could be any single period (e.g., P#January).

The value in the Source POV is stored by the calculation in a temporary account. This is required because the source and destination accounts are typically the same account. Where this is the case, the value in P#[Year] will change as the calculation proceeds from 1 period to the next. Therefore, one has to store the value first to be able to refer to it for all time periods.

It is recommended that variables are set in the calling routine and passed to the Spread function, which define the Destination, Source, Profile, Temp, and Period1 parameters. It is also recommended that the variable names in the calling routine be set to be the same as the Spread function.

Example

The Year value in the account Telephone are allocated across Time Periods using a 4-4-5 quarterly ratio.

The result returned from the SPREAD function is as follows:

```
HS.EXP "A#TempTelephone.C1#[None] = A#Telephone.C1#[None].P#[Year]" (Where
Period.Number = 1)
HS.EXP "A#Telephone.C1#[None] = A#TempTelephone P#January
*                                        E.Globals.A#Profile445.C1#
[None].P#Cur / E.Globals.A#Profile445.C1#[None].P#[Year]
```

Sample Script

The script contains the following information:

- A sample statement written in the calling routine.
- Variables set in the calling routine and passed to the Spread function.
- Variable names in the calling routine set to be the same as the Spread function.

```
Sub Calculate()
Dim Destination
Dim Source
Dim Factor
Dim FactorN
Dim FactorD
Dim Temp
Dim Per
Source = "A#Telephone.C1#[None].P#[Year]"
Destination = "A#Telephone.C1#[None]"
Factor = "A#Factor.C1#[None]"
FactorN = "E#Globals.A#Profile445.C1#[None].P#CUR"
FactorD = "E#Globals.A#Profile445.C1#[None].P#[Year]"
Temp = "A#TempTelephone.C1#[None]"
Per = "January"
Call Spread(Destination,Source,Factor,
FactorN,FactorD,Temp,Per)
End Sub
' Beginning of the Spead function
Sub Spread(Destination,Source,Factor,FactorN,FactorD,Temp,Per)
If HS.Period.Number = 1 Then
HS.Exp Temp & " = " & Source
End If
HS.Clear Factor
HS.EXP Factor & " = " & FactorN & " / " & FactorD
HS.Clear Destination
HS.EXP Destination & " = " & Temp & ".P#" & Per & " * " & Factor
End Sub
```

Units_Rates

This function calculates the product of two source POVs (C = A * B).

Return Value

No return value.

Syntax

`Units_Rates(`*`Destination,Units,Rates`*`)`

Table 25 Syntax of Units_Rates Function

Parameter	Valid Values

Destination	A valid destination POV. That is, a valid combination of Account, ICP and Custom 1-4 members.
Units	A valid source POV. That is, a valid combination of dimension members.
Rates	A valid source POV.

Detailed Description

This function calculates the product of two source POVs (C = A * B). It is recommended that variables are set in the calling routine and passed to the Units_Rates function, which define the Destination, Units and Rates POVs. It is also recommended that the variable names in the calling routine are set to be the same as the Units_Rates function. These recommendations are a suggested best practice approach.

Example

The account Sales will return the value of UnitsSold * Price.

Table 26 Example of Pro_Rata_Ratio Function

	Jan2001	Feb2001	Mar2001
A#UnitsSold	1000	2000	5000
A#Price	1.25	1.00	0.50
Units_Rates("A#UnitsSold",A#Price)	1250	2000	2500

The result returned from the UNITS_RATES function is as follows:

```
HS.EXP "A#Sales = A#UnitsSold * A#Price"
```

Sample Script

The script contains the following information:

- A sample statement written in the calling routine.
- Variables set in the calling routine and passed to the Units_Rates function.
- Variable names in the calling routine set to be the same as the Units_Rates function.

```
Sub Calculate()
Dim Destination
Dim Units
Dim Rates
Destination = "A#Sales"
Units = "A#UnitsSold"
Rates = "A#Price"
Call Units_Rates(Destination,Units,Rates)
End Sub
' Beginning of the Units_Rates function
Sub Units_Rates(Destination,Units,Rates)
HS.EXP Destination & " = " & Units & " * " & Rates
```

```
End Sub
```