

# **Informix DataStage**

## **Developer's Guide**

Version 3.6  
October 1999  
Part No. 000-6466

Published by Informix® Press

Informix Corporation  
4100 Bohannon Drive  
Menlo Park, CA 94025-1032

© 1999 Informix Corporation. All rights reserved. The following are trademarks of Informix Corporation or its affiliates, one or more of which may be registered in the United States or other jurisdictions:

Answers OnLine™; C-ISAM®; Client SDK™; DataBlade®; Data Director™; Decision Frontier™; Dynamic Scalable Architecture™; Dynamic Server™; Dynamic Server™, Developer Edition™; Dynamic Server™ with Advanced Decision Support Option™; Dynamic Server™ with Extended Parallel Option™; Dynamic Server™ with MetaCube®; Dynamic Server™ with Universal Data Option™; Dynamic Server™ with Web Integration Option™; Dynamic Server™, Workgroup Edition™; Dynamic Virtual Machine™; Enterprise Decision Server™; Formation™; Formation Architect™; Formation Flow Engine™; Frameworks for Business Intelligence™; Frameworks Technology™; Gold Mine Data Access®; i.Reach™; i.Sell™; Illustra®; Informix®; Informix® 4GL; Informix® COM Adapter™; Informix® Informed Decisions™; Informix® InquireSM; Informix® Internet Foundation.2000™; InformixLink®; Informix® Red Brick® Decision Server™; Informix Session Proxy™; Informix® Vista™; InfoShelf™; Interforum™; I-Spy™; Mediazation™; MetaCube®; NewEra™; Office Connect™; ON-Bar™; OnLine Dynamic Server™; OnLine/Secure Dynamic Server™; OpenCase®; Orca™; PaVER™; Red Brick® and Design; Red Brick® Data Mine™; Red Brick® Mine Builder™; Red Brick® Decisionscape™; Red Brick® Ready™; Red Brick Systems®; Regency Support®; Rely on Red BrickSM; RISQL®; Solution DesignSM; STARindex™; STARjoin™; SuperView®; TARGETindex™; TARGETjoin™; The Data Warehouse Company®; Universal Data Warehouse Blueprint™; Universal Database Components™; Universal Web Connect™; ViewPoint®; Visionary™; Web Integration Suite™. The Informix logo is registered with the United States Patent and Trademark Office. The DataBlade logo is registered with the United States Patent and Trademark Office.

#### GOVERNMENT LICENSE RIGHTS

Software and documentation acquired by or for the US Government are provided with rights as follows:

(1) if for civilian agency use, with rights as restricted by vendor's standard license, as prescribed in FAR 12.212;  
(2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce this legend.

# Table of Contents

## **Preface**

Organization of This Manual .....	xiii
Documentation Conventions .....	xv
DataStage Documentation .....	xvi

## **Chapter 1. Introduction**

About Data Warehousing .....	1-1
Operational Databases Versus Data Warehouses .....	1-2
Constructing the Data Warehouse .....	1-2
Defining the Data Warehouse .....	1-3
Data Extraction .....	1-3
Data Aggregation .....	1-3
Data Transformation .....	1-3
Advantages of Data Warehousing .....	1-4
Main Features in DataStage .....	1-4

## **Chapter 2. About DataStage**

How DataStage Is Packaged .....	2-1
Client Components .....	2-1
Server Components .....	2-2
DataStage Projects .....	2-2
DataStage Jobs .....	2-2
DataStage NLS .....	2-4
Character Set Maps and Locales .....	2-4
DataStage Terms and Concepts .....	2-5

## **Chapter 3. Your First DataStage Project**

Setting Up Your Project .....	3-2
Starting the DataStage Manager .....	3-2
Defining Table Definitions .....	3-4
Assigning Data Elements .....	3-6

Creating a Job .....	3-7
Developing a Job .....	3-9
Adding Stages .....	3-9
Linking Stages .....	3-10
Editing the Stages .....	3-12
Editing the UniVerse Stage .....	3-12
Editing the Transformer Stage .....	3-16
Editing the Sequential File Stage .....	3-20
Compiling a Job .....	3-21
Running a Job .....	3-22
Analyzing Your Data Warehouse .....	3-23

## **Chapter 4. Setting Up a Project**

Assessing Your Data .....	4-1
Creating the Data Warehouse .....	4-2
The DataStage Manager .....	4-2
Starting the DataStage Manager .....	4-2
The DataStage Manager Window .....	4-4
Using the DataStage Manager .....	4-7
Choosing an Alternative Project .....	4-8
Customizing the Tools Menu .....	4-9
Exiting the DataStage Manager .....	4-10
Table Definitions .....	4-11
Importing a Table Definition .....	4-11
The Table Definition Dialog Box .....	4-23
Manually Entering a Table Definition .....	4-28
Viewing or Modifying a Table Definition .....	4-32
Using the Data Browser .....	4-33
Stored Procedure Definitions .....	4-35
Importing a Stored Procedure Definition .....	4-35
The Table Definition Dialog Box for Stored Procedures .....	4-38
Manually Entering a Stored Procedure Definition .....	4-39
Viewing or Modifying a Stored Procedure Definition .....	4-42

Data Elements .....	4-43
Creating Data Elements .....	4-44
Assigning Data Elements .....	4-46
Viewing or Editing Data Elements .....	4-47
Built-In Data Elements .....	4-47
External ActiveX (OLE) Functions .....	4-48
Importing External ActiveX (OLE) Functions .....	4-49

## **Chapter 5. Developing a Job**

The DataStage Designer .....	5-1
Starting the DataStage Designer .....	5-1
The DataStage Designer Window .....	5-3
Creating a Job .....	5-8
Opening a Job .....	5-9
Saving a Job .....	5-10
Exiting the DataStage Designer .....	5-11
Stages .....	5-11
Built-In Stages .....	5-11
Plug-In Stages .....	5-12
Developing the Job Design .....	5-12
Adding Stages .....	5-13
Moving Stages .....	5-14
Renaming Stages .....	5-14
Deleting Stages .....	5-15
Links .....	5-15
Linking Stages .....	5-16
Editing Stages .....	5-18
Using the Data Browser .....	5-24
The Job Run Options Dialog Box .....	5-26
Containers .....	5-27
Creating a Container .....	5-28
Adding a Container Stage .....	5-28
Viewing or Modifying a Container .....	5-28
Using Input and Output Stages .....	5-29
Job Properties .....	5-30

Specifying Job Parameters .....	5-33
Parameter Types .....	5-34
Parameter Defaults .....	5-35
Defining Job Parameters .....	5-35
Editing Job Parameters .....	5-36
Deleting Job Parameters .....	5-36
Using Job Parameters .....	5-36
Job Control Routines .....	5-38
Specifying Maps and Locales .....	5-41
Defining Character Set Maps .....	5-41
Defining Data Formats with Locales .....	5-42
Specifying Job Dependencies .....	5-43
Specifying Designer Options .....	5-45

## **Chapter 6. ODBC Stages**

Defining the Connection .....	6-2
ODBC Connection Parameters .....	6-2
Defining Character Set Maps .....	6-4
Defining ODBC Input Data .....	6-5
Using a Generated Query .....	6-8
Using a User-Defined SQL Statement .....	6-8
Using a Stored Procedure .....	6-9
Defining ODBC Output Data .....	6-10
Key Fields .....	6-12
Using a Generated Query .....	6-13
Using a User-Defined SQL Statement .....	6-17
Using a Stored Procedure .....	6-19

## **Chapter 7. UniVerse Stages**

Defining the Connection .....	7-2
UniVerse Connection Parameters .....	7-2
Defining UniVerse Input Data .....	7-4
Using a Generated Query .....	7-6
Using a User-Defined SQL Statement .....	7-7

Defining UniVerse Output Data .....	7-8
Key Fields .....	7-10
Using a Generated Query .....	7-10
Using a User-Defined SQL Statement .....	7-15

## **Chapter 8. UniData Stages**

Defining Character Set Maps .....	8-3
Defining UniData Input Data .....	8-4
Defining UniData Output Data .....	8-5

## **Chapter 9. Hashed File Stages**

Defining Hashed File Input Data .....	9-3
Defining Hashed File Output Data .....	9-4

## **Chapter 10. Sequential File Stages**

Defining Character Set Maps .....	10-3
Defining Sequential File Input Data .....	10-4
Defining Sequential File Output Data .....	10-7

## **Chapter 11. Transformer Stages**

Transformer Editor Components .....	11-2
Toolbar .....	11-2
Link Area .....	11-2
Meta Data Area .....	11-2
Shortcut Menus .....	11-3
Transformer Stage Basic Concepts .....	11-4
Input Links .....	11-4
Output Links .....	11-4
Before-Stage and After-Stage Routines .....	11-5
Editing Transformer Stages .....	11-6
Using Drag and Drop .....	11-6
Specifying the Primary Input Link .....	11-7
Creating and Deleting Columns .....	11-7
Moving Columns Within a Link .....	11-8

Editing Column Meta Data .....	11-8
Defining Output Column Derivations .....	11-8
Defining Input Column Key Expressions .....	11-9
Specifying Before-Stage and After-Stage Subroutines .....	11-10
Defining Constraints and Handling Rejects .....	11-11
The DataStage Expression Editor .....	11-12
Entering Expressions .....	11-13
Completing Variable Names .....	11-14
Validating the Expression .....	11-14
Exiting the Expression Editor .....	11-14
Configuring the Expression Editor .....	11-14
Transforms .....	11-15

## **Chapter 12. Aggregator Stages**

Before-Stage and After-Stage Subroutines .....	12-2
Defining Aggregator Input Data .....	12-3
Defining the Input Column Sort Order .....	12-4
Defining Aggregator Output Data .....	12-5
Aggregating Data .....	12-7
The AGGREGATOR Plug-In .....	12-9

## **Chapter 13. Plug-Ins and Plug-In Stages**

Plug-Ins .....	13-1
Manually Registering a Plug-In Definition .....	13-2
Viewing Plug-In Definition Details .....	13-3
Removing a Registered Plug-In .....	13-5
Packaging a Plug-In .....	13-5
Using a Plug-In .....	13-5
Plug-In Stages .....	13-6
Before-Stage and After-Stage Subroutines .....	13-7
Defining Plug-In Input Data .....	13-8
Defining Plug-In Output Data .....	13-9
Editing Properties .....	13-10
Defining Character Set Maps .....	13-12



## **Chapter 14. BCPLoad Stages**

Overview .....	14-1
Before You Start .....	14-2
Table Definitions .....	14-3
SQL Data Types .....	14-3
The BCPLoad Plug-In Definition .....	14-4
Stage Properties .....	14-4
Using the BCPLoad Stage .....	14-5
Editing the BCPLoad Stage .....	14-6
Using Job Parameters .....	14-6
Defining Character Set Maps .....	14-7
Using Stored Procedures .....	14-7
The BCPLoad Demo .....	14-11

## **Chapter 15. Orabulk Stages**

Using the Orabulk Stage .....	15-1
Renaming Columns and Converting Values .....	15-2
Integrity Constraints .....	15-2
Cleaning and Validating Data .....	15-2
Specifying the Stage Properties .....	15-3
Defining Character Set Maps .....	15-4
Loading the Files into Oracle .....	15-5
Running sqldr .....	15-5
Errors and Bad Rows .....	15-6
Empty and Null Values .....	15-7

## **Chapter 16. Programming in DataStage**

Programming Components .....	16-2
Routines .....	16-2
Functions .....	16-3
Expressions .....	16-4
Subroutines .....	16-4
Macros .....	16-5
The Routine Dialog Box .....	16-5

Creating a Routine .....	16-10
Entering Code .....	16-11
Saving Code .....	16-12
Compiling Code .....	16-12
Testing a Routine .....	16-13
Using Find and Replace .....	16-15
Viewing and Editing a Routine .....	16-17
Copying a Routine .....	16-17
Renaming a Routine .....	16-18

## **Chapter 17. Debugging, Compiling, and Releasing a Job**

The DataStage Debugger .....	17-1
Compiling a Job .....	17-5
Compilation Checks .....	17-6
Successful Compilation .....	17-6
Troubleshooting .....	17-7
Releasing a Job .....	17-7

## **Chapter 18. Reporting and Printing**

Reporting .....	18-1
The Reporting Tool .....	18-1
The Documentation Tool .....	18-4
Printing from the Designer .....	18-9
Producing a Printout .....	18-10
Changing the Printer Setup .....	18-10

## **Chapter 19. Importing, Exporting, and Packaging Jobs**

Using Import .....	19-1
Using Export .....	19-3
Using the Packager Wizard .....	19-6

## **Chapter 20. Using MetaBrokers**

Importing Meta Data .....	20-1
Exporting Meta Data .....	20-5

## **Appendix A. Built-In Transforms and Routines**

Built-In Transforms .....	A-1
Transforms Resulting in String .....	A-2
Transforms Resulting in Date .....	A-2
Transform Resulting in Time .....	A-4
Transform Resulting in TIMESTAMP .....	A-5
Transform Resulting in DATE.TAG .....	A-5
Transforms Resulting in WEEK.TAG .....	A-5
Transforms Resulting in MONTH.TAG .....	A-6
Transforms Resulting in QUARTER.TAG .....	A-6
Transforms Resulting in YEAR.TAG .....	A-7
Built-In Routines .....	A-7
Built-In Before/After Subroutines .....	A-8
Example Transform Functions .....	A-8

## **Appendix B. Editing Grids**

Grids .....	B-1
Navigating in the Grid .....	B-2
Editing in the Grid .....	B-3
Adding Rows .....	B-4
Deleting Rows .....	B-4

## **Appendix C. Troubleshooting**

Cannot Start DataStage Clients .....	C-1
Problems While Working with UniData .....	C-1
Connecting to UniData Databases .....	C-1
Importing UniData Meta Data .....	C-2
Using the UniData Stage .....	C-2
Problems with the Documentation Tool .....	C-2
Installing the Documentation Tool .....	C-2
Using Plug-In Reports .....	C-3
Problems Running Jobs .....	C-3
Job Compiles Successfully but Will Not Run .....	C-3
Job from Previous DataStage Release Will Not Run .....	C-3

Miscellaneous Problems .....	C-3
Turning Grid Lines On and Off .....	C-3
Landscape Printing .....	C-3
Browsing for Directories .....	C-3

## **Index**

# Preface

This manual describes the features of the DataStage Manager and DataStage Designer. It is intended for application developers and system administrators who want to use DataStage to design and develop data warehousing applications.

If you are new to DataStage, read the first three chapters for an overview of data warehousing and the concepts and use of DataStage.

The remaining chapters are organized by tasks you must perform to create a data warehousing application. Please refer to the chapters in order because initial setup is the key to a successful DataStage application.

## Organization of This Manual

This manual contains the following:

Chapter 1 contains an overview of data warehousing and describes how DataStage can aid the development and population of a data warehouse.

Chapter 2 introduces the DataStage client and server components and covers DataStage concepts and terminology.

Chapter 3 describes the minimum steps required to create a DataStage application, using the example data provided.

Chapter 4 describes the DataStage Manager and how to set up a DataStage project, including defining table and stored procedure definitions and creating data elements.

Chapter 5 describes how to develop a DataStage job using the DataStage Designer.

Chapter 6 describes how to extract data from or write data to a database table using an ODBC stage. This chapter also describes an alternative way of aggregating output data.

Chapter 7 describes how to extract data from or write data to a database table using a UniVerse stage. This chapter also describes an alternative way of aggregating output data.

Chapter 8 describes how to extract data from or write data to a local UniData file using the UniData stage.

Chapter 9 describes how to extract data from or write data to a local UniVerse file using the Hashed File stage.

Chapter 10 describes how to extract data from or write data to a sequential file using the Sequential File stage.

Chapter 11 describes how to transform (convert) data using a Transformer stage.

Chapter 12 describes how to aggregate output data using the Aggregator stage.

Chapter 13 introduces plug-ins and describes how to create and use them in a plug-in stage.

Chapter 14 describes how to bulk load data into a Microsoft SQL Server or Sybase database table using the BCPLoad plug-in stage.

Chapter 15 describes how to bulk load data into an Oracle database table using the Orabulk plug-in stage.

Chapter 16 describes how to write and test BASIC routines that are used as before-subroutines and after-subroutines or transform functions.

Chapter 17 describes how to debug, compile, and release a DataStage job.

Chapter 18 covers how to generate reports from the DataStage Manager and Designer, and how to print from the DataStage Manager and Designer.

Chapter 19 describes how to import and export DataStage components between development environments, and how to package jobs for deployment to other systems.

Chapter 20 tells you how to use MetaBrokers to exchange meta data between DataStage and other data warehousing tools.

Appendix A describes the built-in transforms and routines supplied with DataStage.

Appendix B covers how to navigate and edit the grids that appear in many DataStage dialog boxes.

Appendix C gives troubleshooting advice.

# Documentation Conventions

This manual uses the following conventions:

Convention	Usage
<b>Bold</b>	In syntax, bold indicates commands, function names, keywords, and options that must be input exactly as shown. In text, bold indicates keys to press, function names, and menu selections.
UPPERCASE	In syntax, uppercase indicates BASIC statements and functions and SQL statements and keywords.
<i>Italic</i>	In syntax, italic indicates information that you supply. In text, italic also indicates operating system commands and options, filenames, and pathnames.
Courier	Courier indicates examples of source code and system output.
<b>Courier Bold</b>	In examples, courier bold indicates characters that the user types or keys the user presses (for example, <Return>).
[ ]	Brackets enclose optional items. Do not type the brackets unless indicated.
{ }	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
itemA   itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
...	Three periods indicate that more of the same type of item can optionally follow.
➤	A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose <b>File</b> ➤ <b>Exit</b> ” means you should choose <b>File</b> from the menu bar, then choose <b>Exit</b> from the File pull-down menu.
This line ➔ continues	The continuation character is used in source code examples to indicate a line that is too long to fit on the page, but must be entered as a single line on screen.

The following conventions are also used:

- Syntax definitions and examples are indented for ease in reading.
- All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.

- Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

## DataStage Documentation

DataStage documentation includes the following:

***DataStage Developer's Guide:*** This guide describes the DataStage Manager and Designer, and how to create, design, and develop a DataStage application.

***DataStage Operator's Guide:*** This guide describes the DataStage Director and how to validate, schedule, run, and monitor DataStage applications.

***DataStage Administrator's Guide:*** This guide describes DataStage setup, routine housekeeping, and administration.

These guides are also available online in PDF format. You can read them with the Adobe Acrobat Reader supplied with DataStage. See *DataStage Installation Instructions* in the DataStage CD jewel case for details on installing the manuals and the Adobe Acrobat Reader.



# 1

## Introduction

This chapter is an overview of data warehousing and Informix DataStage.

The last few years have seen the continued growth of IT (information technology) and the requirement of organizations to make better use of the data they have at their disposal. This involves analyzing data in active databases and comparing it with data in archive systems.

Although offering the advantage of a competitive edge, the cost of consolidating data into a data mart or data warehouse was high. It also required the use of data warehousing tools from a number of vendors and the skill to create a data warehouse.

Developing a data warehouse or data mart involves design of the data warehouse and development of operational processes to populate and maintain it. In addition to the initial setup, you must be able to handle on-going evolution to accommodate new data sources, processing, and goals.

DataStage from Ardent simplifies the data warehousing process. It is an integrated product that supports extraction of the source data, cleansing, decoding, transformation, integration, aggregation, and loading of target databases.

Although primarily aimed at data warehousing environments, DataStage can also be used in any data handling, data migration, or data reengineering projects.

### **About Data Warehousing**

The aim of data warehousing is to make more effective use of the data available in an organization and to aid decision-making processes.

A data warehouse is a central integrated database containing data from all the operational sources and archive systems in an organization. It contains a copy of transaction data specifically structured for query analysis. This database can be accessed by all users, ensuring that each group in an organization is accessing valuable, stable data.

A data warehouse is a “snapshot” of the operational databases combined with data from archives. The data warehouse can be created or updated at any time, with minimum disruption to operational systems. Any number of analyses can be performed on the data, which would otherwise be impractical on the operational sources.

## Operational Databases Versus Data Warehouses

Operational databases are usually accessed by many concurrent users. The data in the database changes quickly and often. It is very difficult to obtain an accurate picture of the contents of the database at any one time.

Because operational databases are task oriented, for example, stock inventory systems, they are likely to contain “dirty” data. The high throughput of data into operational databases makes it difficult to trap mistakes or incomplete entries. However, you can cleanse data before loading it into a data warehouse, ensuring that you store only “good” complete records.

## Constructing the Data Warehouse

A data warehouse is created by extracting data from one or more operational databases. The data is transformed to eliminate inconsistencies, aggregated to summarize data, and loaded into the data warehouse. The end result is a dedicated database which contains stable, nonvolatile, integrated data. This data also represents a number of time variants (for example, daily, weekly, or monthly values), allowing the user to analyze trends in the data.

The data in a data warehouse is classified based on the subjects of interest to the organization. For a bank, these subjects may be customer, account number, and transaction details. For a retailer, these may include product, price, quantity sold, and order number.

Each data warehouse includes detailed data. However, where only a portion of this detailed data is required, a data mart may be more suitable. A *data mart* is generated from the data contained in the data warehouse and contains focused data that is frequently accessed or summarized, for example, sales or marketing data.

The person who constructs the data warehouse must know the needs of users who will use the data warehouse or data marts. This means knowing the data contained in each operational database and how each database is related (if at all).

## **Defining the Data Warehouse**

Defining the warehouse is one of the first steps in creating a data warehouse. The definition describes the content of the data warehouse by specifying the data elements and any transforms (conversions) required before the data is stored. The definition of the data warehouse is described in terms of meta data. Meta data is data about the data you are handling – typically a set of column definitions describing the structure of the data.

Meta data can be created using the schemas or subschemas that are used to define the operational databases. Although meta data can be difficult to define and be a time-consuming process, it holds the key to a successful data warehouse.

## **Data Extraction**

The data in operational or archive systems is the primary source of data for the data warehouse. Operational databases can be indexed files, networked databases, or relational database systems. Data extraction is the process used to obtain data from operational sources, archives, and external data sources.

## **Data Aggregation**

An operational data source usually contains records of individual transactions such as product sales. If the user of a data warehouse only needs a summed total, you can reduce records to a more manageable number by aggregating the data.

The summed (aggregated) total is stored in the data warehouse. Because the number of records stored in the data warehouse is greatly reduced, it is easier for the end user to browse and analyze the data.

## **Data Transformation**

Because the data in a data warehouse comes from many sources, the data may be in different formats or be inconsistent. Transformation is the process that converts data to a required definition and value.

Data is transformed using routines based on a transformation rule, for example, product codes can be mapped to a common format using a transformation rule that applies only to product codes.

After data has been transformed it can be loaded into the data warehouse in a recognized and required format.

## Advantages of Data Warehousing

A data warehousing strategy provides the following advantages:

- Capitalizes on the potential value of the organization's information
- Improves the quality and accessibility of data
- Combines valuable archive data with the latest data in operational sources
- Increases the amount of information available to users
- Reduces the requirement of users to access operational data
- Reduces the strain on IT departments, as they can produce one database to serve all user groups
- Allows new reports and studies to be introduced without disrupting operational systems
- Promotes users to be self sufficient

## Main Features in DataStage

DataStage has the following features to aid the design and processing required to build a data warehouse:

- Uses graphical design tools. With simple point and click techniques you can draw a scheme to represent your processing requirements.
- Extracts data from any number or types of database.
- Handles all the meta data definitions required to define your data warehouse. You can view and modify the table definitions at any point during the design of your application.
- Aggregates data. You can modify SQL SELECT statements used to extract data.
- Transforms data. DataStage has a set of predefined transforms and functions you can use to convert your data. You can easily extend the functionality by defining your own transforms to use.
- Loads the data warehouse.

DataStage consists of a number of client and server components. For more information, see "Client Components" on page 2-1 and "Server Components" on page 2-2.

# 2

## About DataStage

DataStage is a tool set for designing, developing, and running applications that populate one or more tables in a data warehouse or data mart.

This chapter provides:

- A basic introduction to DataStage components and concepts
- A description of the DataStage terms used throughout this manual

### How DataStage Is Packaged

DataStage consists of client and server components. The client and server components installed depend on the edition of DataStage you have purchased. DataStage is packaged in two ways:

- **Developer's Edition.** Used by developers to design, develop, and create executable DataStage jobs. Contains all the client and server components described next.
- **Operator's Edition.** Used by operators to validate, schedule, run, and monitor executable DataStage jobs. Contains the DataStage Director and Server components. The DataStage Director and the operator's role are described in *DataStage Operator's Guide*.

### Client Components

DataStage has four client components which are installed on any PC running Windows 98 or Windows NT 4.0 or later:

- **DataStage Designer.** A design interface used to create DataStage applications (known as jobs). Each job specifies the data sources, the transforms required, and the destination of the data. Jobs are compiled to create executables that are scheduled by the Director and run by the Server.

- **DataStage Director.** A user interface used to validate, schedule, run, and monitor DataStage jobs.
- **DataStage Manager.** A user interface used to view and edit the contents of the Repository.
- **DataStage Administrator.** A user interface used to set up DataStage users, create and move projects, and set up purging criteria.

## Server Components

There are three server components:

- **Repository.** A central store that contains all the information required to build a data mart or data warehouse.
- **DataStage Server.** Runs executable jobs that extract, transform, and load data into a data warehouse.
- **DataStage Package Installer.** A user interface used to install packaged DataStage jobs and plug-ins.

## DataStage Projects

You always enter DataStage through a DataStage project. When you start a DataStage client you are prompted to attach to a project. Each project contains:

- DataStage jobs.
- Built-in components. These are predefined components used in a job.
- User-defined components. These are customized components created using the DataStage Manager. Each user-defined component performs a specific task in a job.

A complete project may contain several jobs and user-defined components.

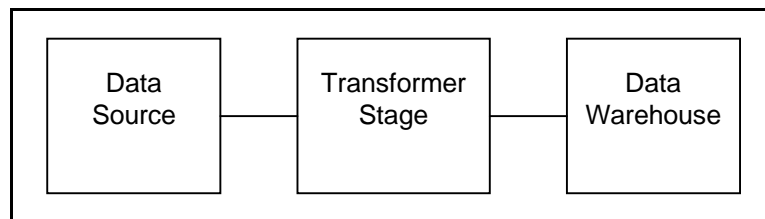
## DataStage Jobs

DataStage jobs consist of individual stages. Each stage describes a particular database or process. For example, one stage may extract data from a data source, while another transforms it. Stages are added to a job and linked together using the Designer.

There are two types of stage:

- **Built-in stages.** Supplied with DataStage and used for extracting, aggregating, transforming, or writing data.
- **Plug-in stages.** Additional stages that can be installed in DataStage to perform specialized tasks that the built-in stages do not support.

The following diagram represents one of the simplest jobs you could have: a data source, a transformer (conversion) stage, and the final database. The links between the stages represent the flow of data into or out of a stage.



You must specify the data you want at each stage, and how it is handled. For example, do you want all the columns in the source data, or only a select few? Should the data be aggregated or converted before being passed on to the next stage?

Project-wide data properties determine the possible operations on your data. These properties are available to all the jobs in your project and consist of:

- **Table definitions.** These specify the data you want. Each table definition contains:
  - Information about the table or file that holds the data records.
  - A description of the individual columns.
- **Data elements.** Each data element describes one type of data that can be stored in a column. The data element associated with a column defines the operations that can be carried out on that column. DataStage has numerous predefined data elements representing commonly required data types (such as date, time, number, and string). You can also define your own special data elements.
- **Transforms.** These convert and cleanse your data by transforming it into a format you want to save and use in your final warehouse.

You can use DataStage in conjunction with Ardent MetaBrokers in order to exchange meta data with other data warehousing tools. You might, for example, import table definitions from a data modelling tool.

An executable job is created when you compile a job design. When the job is run, the processing stages described in the job design are performed using the data properties you defined. Executable jobs can be packaged for use on other DataStage systems.

## DataStage NLS

DataStage has built-in National Language Support (NLS). With NLS installed, DataStage can do the following:

- Process data in a wide range of languages.
- Accept data in any character set into most DataStage fields.
- Use local formats for dates, times, and money.
- Sort data according to local rules.
- Convert data between different encodings of the same language (for example, for Japanese it can convert JIS to EUC).

DataStage NLS is optionally installed as part of the DataStage server. If NLS is installed, various extra features (such as dialog box pages and drop-down lists) appear in the product. If NLS is not installed, these features do not appear.

Using NLS, the DataStage server engine holds data in UNICODE format. This is an international standard character set that contains nearly all the characters used in languages around the world. DataStage maps data to or from UNICODE format as required.

## Character Set Maps and Locales

Each DataStage project has a language assigned to it during installation. This equates to one or more character set maps and locales which support that language. One map and one locale are assigned as project defaults.

- The maps define the character sets that the project can use.
- The locales define the local formats for dates, times, sorting order, and so on that the project can use.

The DataStage client and server components also have maps assigned to them during installation to ensure that data is transferred between them in the correct character set. For more information, see *DataStage Administrator's Guide*.



When you design a DataStage job, you can override the project default map at several levels:

- For a job
- For a stage within a job
- For a column within a stage (for Sequential, ODBC, and generic plug-in stages)
- For transforms and routines used to manipulate data within a stage
- For imported meta data and table definitions

The locale and character set information becomes an integral part of the job. When you package and release a job, the NLS support can be used on another system, provided that the correct maps and locales are installed and loaded.

## DataStage Terms and Concepts

The following terms are used in DataStage:

<b>Term</b>	<b>Description</b>
administrator	The person who is responsible for the maintenance and configuration of DataStage, and for DataStage users.
after-job subroutine	A routine that is executed after a job runs.
after-stage subroutine	A routine that is executed after a stage processes data.
Aggregator stage	A stage type that computes totals or other functions of sets of data.
BCPLoad stage	A plug-in stage supplied with DataStage that bulk loads data into a Microsoft SQL Server or Sybase table.
before-job subroutine	A routine that is executed before a job is run.
before-stage subroutine	A routine that is executed before a stage processes any data.
built-in data elements	There are two types of built-in data elements: those that represent the base types used by DataStage during processing and those that describe different date/time formats.
built-in transforms	The transforms supplied with DataStage. See Appendix A for a complete list.

<b>Term</b>	<b>Description</b>
column definition	Defines the columns contained in a data table. Includes the column name and the type of data contained in the column.
container	A group of stages and links in a job design.
Container stage	A built-in stage type that represents a group of stages and links in a job design.
custom transform	A transform function defined by the DataStage developer.
data element	A specification that describes the type of data in a column and how the data is converted.
Data Browser	A tool used from within the DataStage Manager or DataStage Designer to view the content of a table or file.
DataStage Administrator	A tool used to configure DataStage projects and users. For more details, see <i>DataStage Administrator's Guide</i> .
DataStage Designer	A graphical design tool used by the developer to design and develop a DataStage job.
DataStage Director	A tool used by the operator to run and monitor DataStage jobs.
DataStage Manager	A tool used to view and edit definitions in the Repository.
DataStage Package Installer	A tool used to install packaged DataStage jobs and plug-ins.
developer	The person designing and developing DataStage jobs.
Expression Editor	An interactive editor that helps you to enter correct expressions into a Transformer stage in a DataStage job design.
Hashed File stage	A stage that extracts data from or loads data into a database that contains hashed files.
job	A collection of linked stages, data elements, and transforms that define how to extract, cleanse, transform, integrate, and load data into a target database. A job can be compiled to produce an executable.
job control routine	A routine that is used to create a controlling job, which invokes and runs other jobs.

<b>Term</b>	<b>Description</b>
meta data	Data about data, for example, a table definition describing columns in which data is structured.
MetaBroker	A tool that allows you to exchange meta data between DataStage and other data warehousing tools.
NLS	National Language Support. With NLS enabled, DataStage can support the handling of data in a variety of character sets.
normalization	The conversion of records in NF <sup>2</sup> (nonfirst-normal form) format, containing multivalued data, into one or more 1NF (first normal form) rows.
ODBC stage	A stage that extracts data from or loads data into a database that implements the industry standard Open Database Connectivity API. Used to represent a data source, an aggregation step, or a target data table.
operator	The person scheduling and monitoring DataStage jobs.
Orabulk stage	A plug-in stage supplied with DataStage that bulk loads data into an Oracle database table.
plug-in	A definition for a plug-in stage. For more information, see Chapter 13, "Plug-Ins and Plug-In Stages."
plug-in stage	A stage that performs specific processing that is not supported by the Aggregator, Hashed File, ODBC, UniVerse, UniData, Sequential File, and Transformer stages.
Repository	A DataStage area where projects and jobs are stored as well as definitions for all standard and user-defined data elements, transforms, and stages.
Sequential File stage	A stage that extracts data from, or writes data to, a text file.
source	A source in DataStage terms means any database, whether you are extracting data from it or writing data to it.
stage	A component that represents a data source, a processing step, or the data mart in a DataStage job.

<b>Term</b>	<b>Description</b>
table definition	A definition describing the data you want including information about the data table and the columns associated with it. Also referred to as meta data.
transform function	A function that takes one value and computes another value from it.
Transformer editor	A graphical interface for editing Transformer stages.
Transformer stage	A stage where data is transformed (converted) using transform functions.
UniData stage	A stage that extracts data from or loads data into a UniData database. Used to represent a data source or a target data table.
UniVerse stage	A stage that extracts data from or loads data into a UniVerse database using SQL. Used to represent a data source, an aggregation step, or a target data table.

# 3

## Your First DataStage Project

This chapter describes the steps you need to follow to create your first data warehouse, using the sample data provided. The example uses a UniVerse table called *EXAMPLE1*, which is automatically copied into your DataStage project during server installation.

*EXAMPLE1* represents an SQL table from a wholesaler who deals in car parts. It contains details of the wheels they have in stock. There are approximately 255 rows of data and four columns:

- **CODE.** The product code for each type of wheel.
- **PRODUCT.** A text description of each type of wheel.
- **DATE.** The date new wheels arrived in stock (given in terms of year, month, and day).
- **QTY.** The number of wheels in stock.

The aim of this example is to develop and run a DataStage job that:

- Extracts the data from the file.
- Converts (transforms) the data in the DATE column from a complete date (YYYY-MM-DD) stored in internal data format, to a year and month (YYYY-MM) stored as a string.
- Loads data from the DATE, CODE, and QTY columns into a data warehouse. The data warehouse is a sequential file that is created when you run the job.

To load a data mart or data warehouse, you must do the following:

- Set up your project
- Create a job

- Develop the job
- Edit the stages in the job
- Compile the job
- Run the job

This chapter describes the minimum tasks required to create a DataStage job. In the example, you will use the built-in settings and options supplied with DataStage. However, because DataStage allows you to customize and extend the built-in functionality provided, it is possible to perform additional processing at each step. Where this is possible, additional procedures are listed under a section called *Advanced Procedures*. These advanced procedures are discussed in detail in subsequent chapters.

## Setting Up Your Project

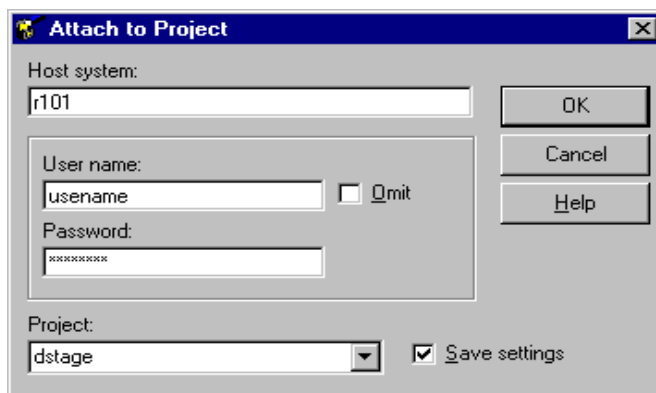
Before you create any DataStage jobs, you must set up your project by entering information about your data. This includes the name and location of the tables or files holding your data and a definition of the columns they contain.

Information is stored in table definitions in the Repository and is entered using the DataStage Manager. The easiest way to enter a table definition is to import directly from the source data.

In this example, you need a table definition for *EXAMPLE1*.

## Starting the DataStage Manager

To start the DataStage Manager, choose **Start > Programs > Ardent DataStage > DataStage Manager**. The Attach to Project dialog box appears:



This dialog box appears when you start the DataStage Manager, Designer, or Director client components from the DataStage program folder. In all cases, you must attach to a project by entering your logon details.

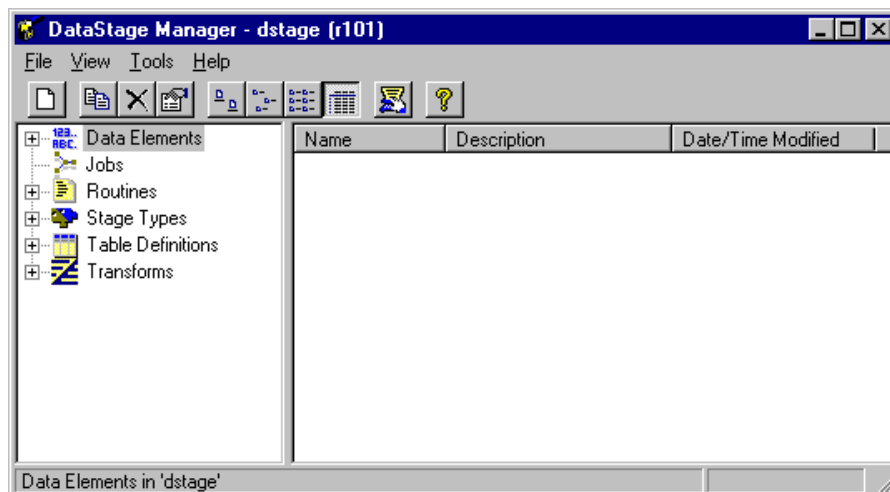
**Note:** The program group may be called something other than Ardent DataStage, depending on how DataStage was installed.

To connect to a project:

1. Enter the name of your host in the **Host system** field. This is the name of the system where the DataStage Server components are installed.
2. Enter your user name in the **User name** field. This is your user name on the server system.
3. Enter your password in the **Password** field.

**Note:** If you are connecting to the server via LAN Manager, you can select the **Omit** check box. The **User name** and **Password** fields gray out and you log on to the server using your Windows NT Domain account details.

4. Choose the project to connect to from the Project drop-down list box. This list box displays all the projects installed on your DataStage server. Choose your project from the list box. At this point, you may only have one project installed on your system and this is displayed by default.
5. Select the **Save settings** check box to save your logon settings.
6. Click **OK**. The DataStage Manager window appears:



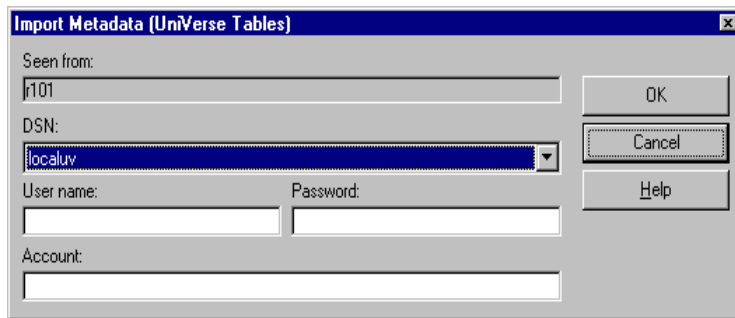
## Defining Table Definitions

If you have a hashed file, UniVerse table, UniData, or ODBC data source, the quickest and simplest way to specify a table definition is to import it directly from your data source or data warehouse. In this example, you must specify a table definition for *EXAMPLE1*.

## Importing a Table Definition

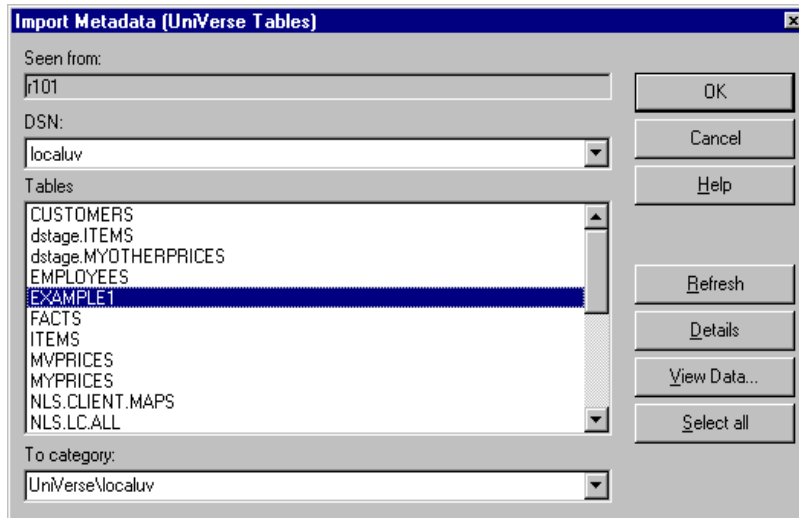
The following steps describe how to import a table definition for *EXAMPLE1*:

1. Choose **Tools** ► **Import** ► **UniVerse Table Definitions...** . The Import Metadata (UniVerse Tables) dialog box appears:



2. Choose **localuv** from the DSN drop-down list box.
3. Click **OK**. The updated Import Metadata (UniVerse Tables) dialog box displays all the files for the chosen data source name:





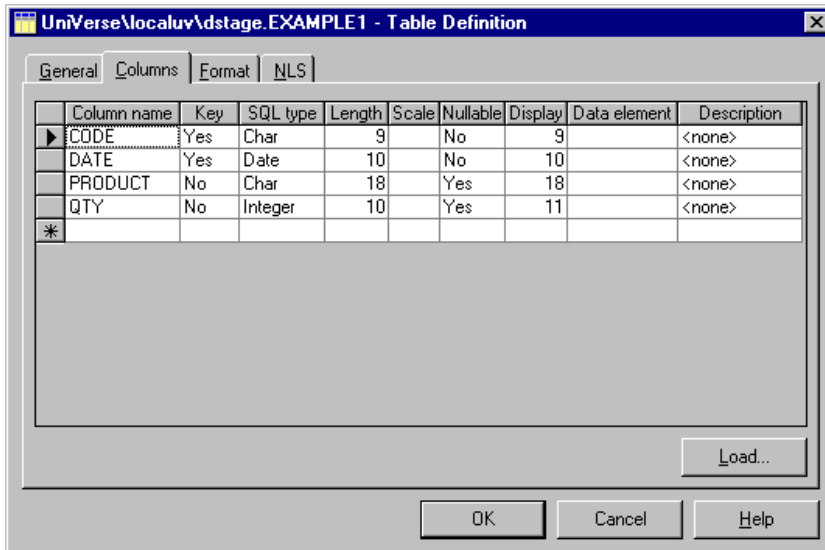
**Note:** The screen shot shows an example of tables found under *localuv*. Your system may contain different files to the ones shown here.

4. Select ***project.EXAMPLE1*** from the Tables list box, where *project* is the name of your DataStage project.
5. Click **OK**. The column information from *EXAMPLE1* is imported into DataStage. A table definition is created and is stored under the **Table Definitions ► UniVerse ► localuv** branch in the Repository. The updated DataStage Manager window displays the new table definition entry in the display area.

To view the new table definition, double-click the ***project.EXAMPLE1*** entry in the display area. The Table Definition dialog box appears.

This dialog box has four pages. Click the tabs to display each page. The **General** page contains information about where the data is found and when the definition was created.

The **Columns** page contains information about the columns in the data source table. You should see the following columns for *project.EXAMPLE1*:



The **Format** page contains information describing how the data would be formatted when written to a sequential file. You do not need to edit this page.

The **NLS** page shows the current character set map for the table definitions. The map defines the character set that the data is in. You do not need to edit this page.

### **Advanced Procedures**

To manually enter table definitions, see “Manually Entering a Table Definition” on page 4-28.

## **Assigning Data Elements**

A DataStage data element describes more precisely the kind of data that can appear in a given column. When you import a table definition, no DataStage data elements are set. You can apply a DataStage data element to give more meaning to a particular column and to specify a transform to use. Transforms are defined such that they convert one data element to another.

In this example, to transform the data in the DATE column, you assign a DataStage data element. Initially you assign the Date type. (Later, in the Transformer stage, you will assign a MONTH.TAG data element to the output column to specify that the transform produces a string of the format YYYY-MM.)

**Note:** If the data in the other columns required transforming, you could assign DataStage data elements to these columns too.

To assign a data element:

1. Click the **Data element** cell for the **DATE** row in the Columns grid.
2. Choose **Date** from the Data element drop-down list box.
3. Click **OK** to save the table definition and to close the Table Definition dialog box.

### ***Advanced Procedures***

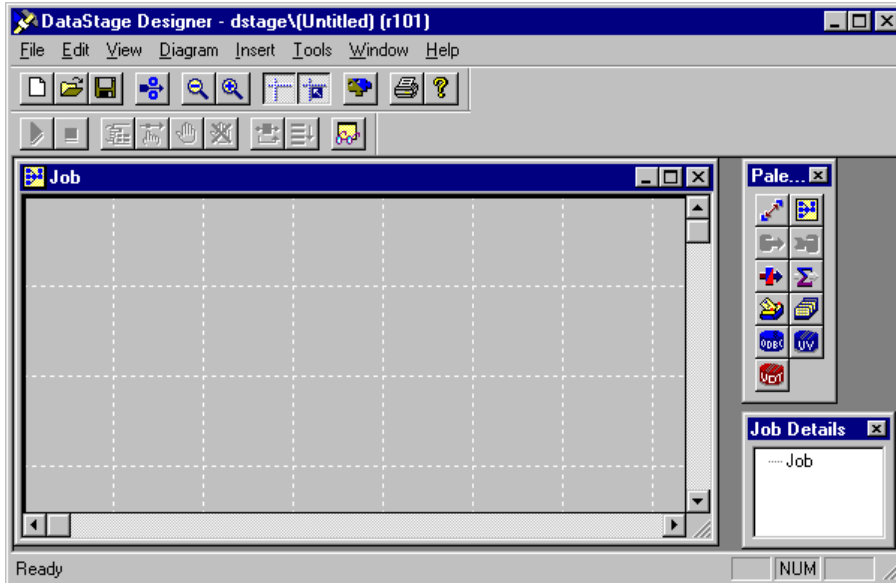
For more advanced procedures, see the following topics in Chapter 4:

- “Creating Data Elements” on page 4-44
- “Assigning Data Elements” on page 4-46

## **Creating a Job**

When a DataStage project is installed, it is empty and you must create the jobs you need. Each DataStage job can load one or more data tables in the final data warehouse. The number of jobs you have in a project depends on your data sources and how often you want to extract data or load the data warehouse.

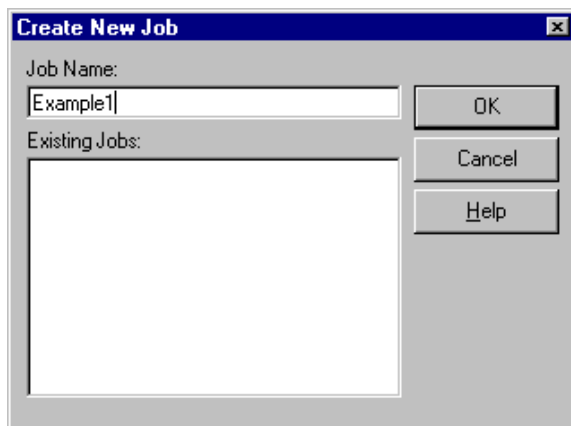
Jobs are created using the DataStage Designer. Start the DataStage Designer from the Manager by choosing **Tools ► Run Designer**. The DataStage Designer window appears:



When you start the DataStage Designer, a new job is created automatically which you can save and edit.

To save the job:

1. Choose **File** ► **Save Job**. The Create New Job dialog box appears:



2. Enter **Example1** in the **Job Name** field.

3. Click **OK** to save the job. The updated DataStage Designer window displays the name of the saved job.

## Developing a Job

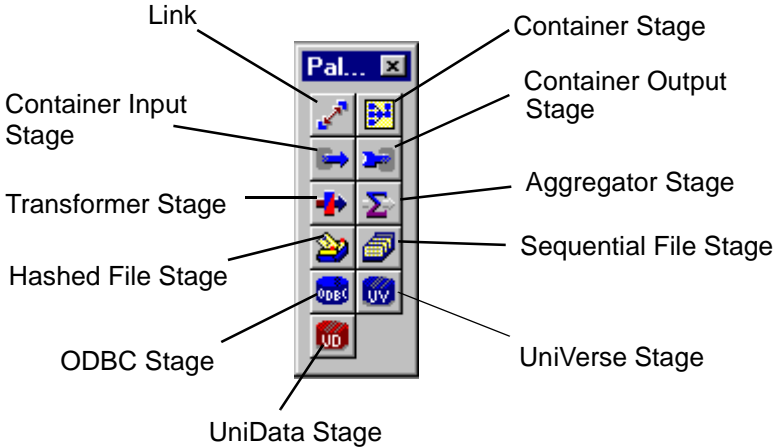
Jobs are designed and developed using the Designer. The job design is developed in the Diagram window (the one with grid lines). Each data source, the data warehouse, and each processing step is represented by a stage in the job design. The stages are linked together to show the flow of data.

This example requires three stages:

- A UniVerse stage to represent *EXAMPLE1* (the data source)
- A Transformer stage to convert the data in the DATE column from a YYYY-MM-DD date in internal date format to a string giving just year and month (YYYY-MM)
- A Sequential File stage to represent the file created at run time (the data warehouse in this example)

## Adding Stages

Stages are added using the tool palette. This palette contains icons that represent the components you can add to a job.



To add a stage:

1. Click the stage icon on the tool palette that represents the stage type you want to add.
2. Click in the Diagram window where you want the stage to be positioned. The stage appears in the Diagram window as a square.

We recommend that you position your stages as follows:

- Data sources on the left
- Data warehouse on the right
- Transformer stage in the center

When you add stages, they are automatically assigned default names. These names are based on the type of stage and the number of the item in the Diagram window. You can use the default names in the example.

Once all the stages are in place, you can link them together to show the flow of data.

## Linking Stages

You need to add two links:

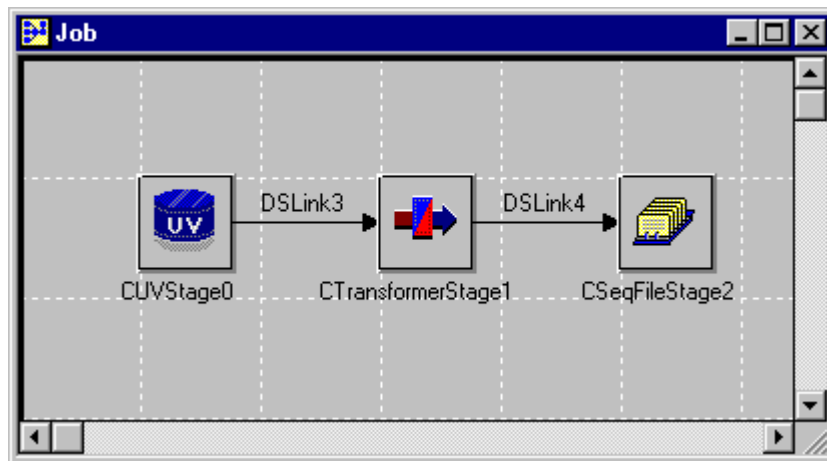
- One between the UniVerse and Transformer stages
- One between the Transformer and Sequential File stages

Links are always made in the direction the data will flow, that is, usually left to right. When you add links, they are assigned default names. You can use the default names in the example.

To add a link:

1. Click the **Link** icon on the tool palette.
2. Click the first stage and drag the link to the Transformer stage. The link appears as soon as you release the mouse button.
3. Click the **Link** icon on the tool palette again.

4. Click the Transformer stage and drag the link to the Sequential File stage. The following screen shows how the Diagram window looks when you have added the stages and links:



5. Save the job design by choosing **File ► Save Job**.  
Keep the Designer open as you will need it for the next step.

### ***Advanced Procedures***

For more advanced procedures, see the following topics in Chapter 5:

- “Plug-In Stages” on page 5-12
- “Moving Stages” on page 5-14
- “Renaming Stages” on page 5-14
- “Deleting Stages” on page 5-15
- “Containers” on page 5-27
- “Specifying Job Parameters” on page 5-33

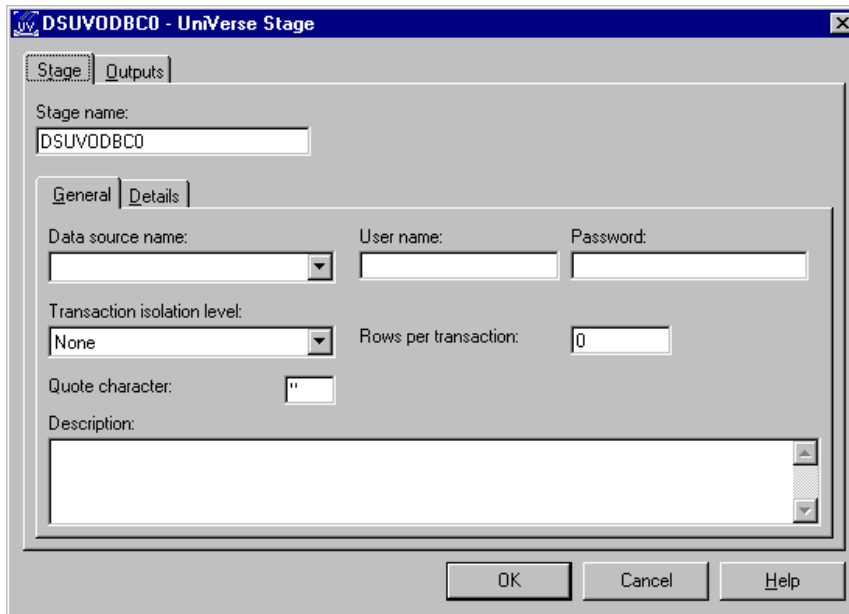
## Editing the Stages

Your job design currently displays the stages and the links between them. You must edit each stage in the job to specify the data to use and what to do with it. Stages are edited in the job design by double-clicking each stage in turn. Each stage type has its own editor.

### Editing the UniVerse Stage

The data source (*EXAMPLE1*) is represented by a UniVerse stage. You must specify the data you want to extract from this file by editing the stage.

Double-click the stage to edit it. The UniVerse Stage dialog box appears:



This dialog box has two pages:

- **Stage.** Displayed by default. This page contains the name of the stage you are editing. The **General** page specifies where the file is found and the connection type.
- **Outputs.** Contains information describing the data flowing from the stage. You edit this page to describe the data you want to extract from the file. In this example, the output from this stage goes to the Transformer stage.

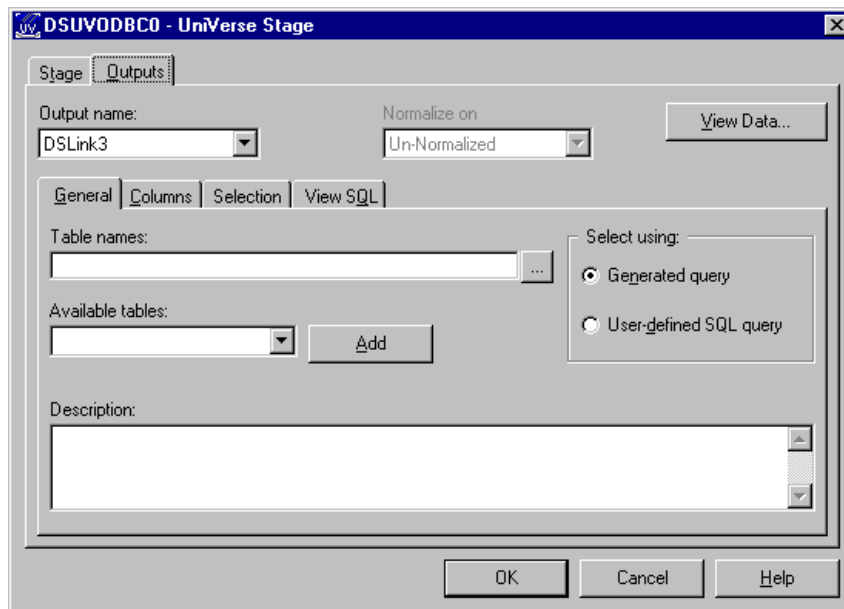


To edit the UniVerse stage:

1. Check that you are displaying the **General** page on the **Stage** page. Choose **localuv** from the Data source name drop-down list box. **localuv** is where **EXAMPLE1** is copied to during installation.

The remaining parameters on the **General** and **Details** pages are used to enter logon details and describe where to find the file. Because **EXAMPLE1** is installed in **localuv**, you do not have to complete these fields, which are disabled.

2. Click the **Outputs** tab. The **Outputs** page appears:



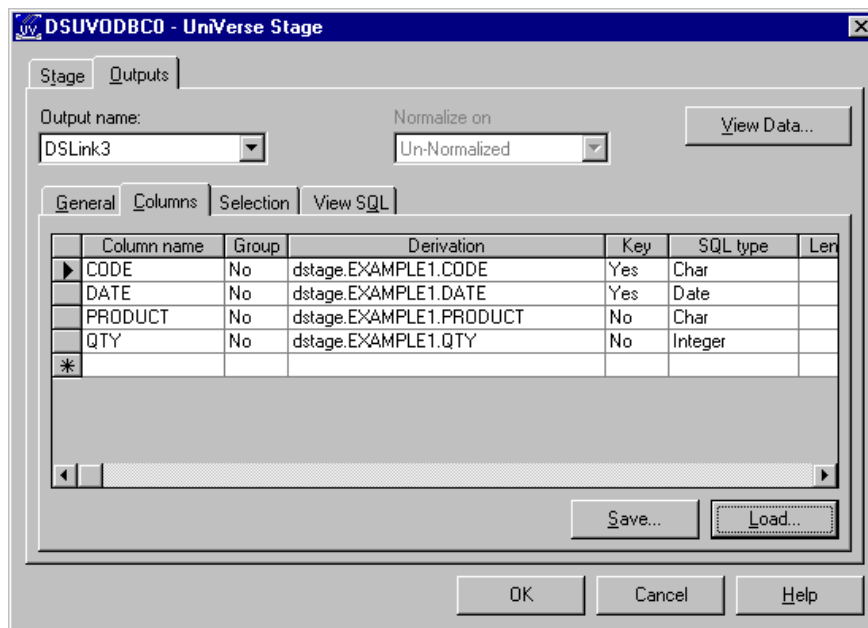
The **Outputs** page contains the name of the link the data flows along and the following four pages:

- **General.** Contains the name of the table to use and an optional description of the link.
- **Columns.** Contains information about the columns in the table.
- **Selection.** Used to enter an optional SQL SELECT clause (an *Advanced* procedure).
- **View SQL.** Displays the SQL SELECT statement used to extract the data.

3. Choose **dstage.EXAMPLE1** from the Available tables drop-down list box.
4. Click **Add** to add **dstage.EXAMPLE1** to the **Table names** field.
5. Click the **Columns** tab. The **Columns** page appears at the front of the dialog box.

You must specify the columns contained in the file you want to use. Because the column definitions are stored in a table definition in the Repository, you can load them directly.

6. Click **Load...** . The Table Definitions window appears with the **UniVerse** ► **localuv** branch highlighted.
7. Select **dstage.EXAMPLE1**.
8. Click **OK**. The column definitions specified in the table definition are copied to the stage. The **Columns** page contains definitions for the four columns in **EXAMPLE1**:



9. You can use the Data Browser to view the actual data that is to be output from the UniVerse stage. Click the **View Data...** button to invoke the Data Browser window.

CODE	DATE	PRODUCT	QTY
789ZZZ12	1981-10-0	155x13 Standard X	18
789ZZZ12	1981-11-2	155x13 Standard X	6
789ZZZ12	1983-01-0	155x13 Standard X	18
789ZZZ12	1983-09-0	155x13 Standard X	20
789ZZZ12	1983-10-1	155x13 Standard X	26
789ZZZ12	1984-02-0	155x13 Standard X	22
789ZZZ12	1984-09-1	155x13 Standard X	22
123EEE44	1986-03-1	165x13 Standard X	22
123EEE44	1986-04-1	165x13 Standard X	4
123EEE44	1987-04-0	165x13 Standard X	29
123EEE44	1988-03-1	165x13 Standard X	13
123EEE44	1988-09-2	165x13 Standard X	6
222AAA71	1988-10-1	205x15 Pro-grip Z	24
222AAA71	1988-10-3	205x15 Pro-grip Z	12
222AAA71	1989-03-1	205x15 Pro-grip Z	17

10. Click **OK** to save the stage edits and to close the UniVerse Stage dialog box.

11. Choose **File ► Save Job** to save your job design so far.

**Note:** Column definitions are attached to a link. You can view or edit them at either end of the link. If you change them in a stage at one end of the link, the changes are automatically seen in the stage at the other end of the link. This is how column definitions are propagated through all the stages in a DataStage job, so the column definitions you loaded into the UniVerse stage are viewed when you edit the Transformer stage.

### Advanced Procedures

- To select only some of the data from the table using a WHERE clause, see “Using a WHERE Clause” on page 6-14 and page 7-12.
- To use SQL SELECT statements to aggregate data (for ODBC and UniVerse stages only), see “Aggregating Data” on page 6-16 and page 7-13.

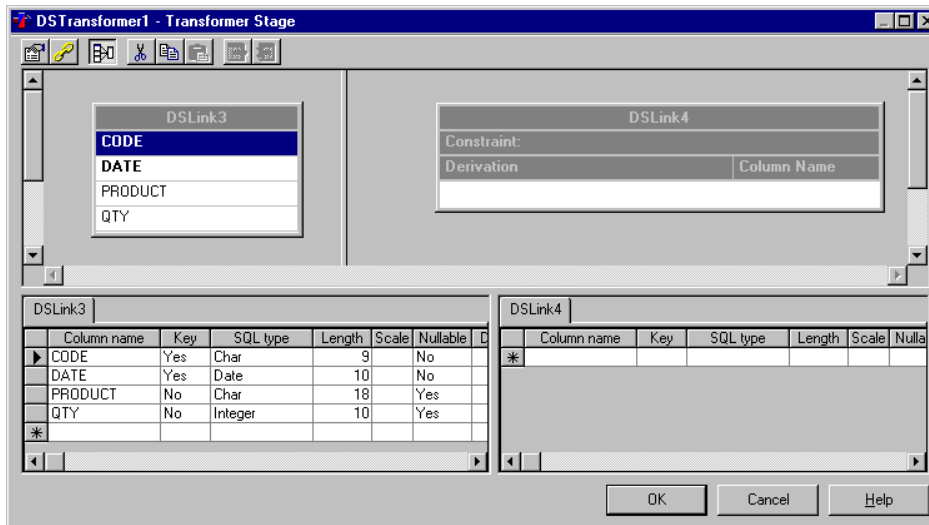
## Editing the Transformer Stage

The Transformer stage performs any data conversion required before the data is output to another stage in the job design. In this example, the Transformer stage is used to convert the data in the DATE column from a YYYY-MM-DD date in internal date format to a string giving just the year and month (YYYY-MM).

There are two links in this stage:

- The input from the data source (*EXAMPLE1*)
- The output to the Sequential File stage

Double-click the stage to edit it. The Transformer Editor appears:

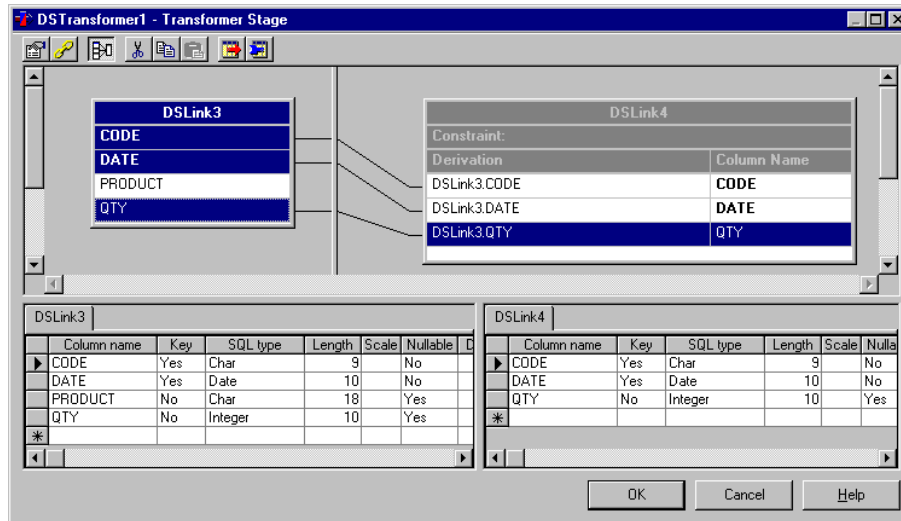


Input columns are shown on the left, output columns on the right. The upper panes show the columns together with derivation details, the lower panes show the column meta data. In this case, input columns have already been defined for input link DLink3. No output columns have been defined for output link DLink4, so the right panes are blank.

The next steps are to define the columns that will be output by the Transformer stage, and to specify the transform that will enable the stage to convert the type and format of dates before they are output.

1. Working in the upper-left pane of the Transformer Editor, select the input columns that you want to derive output columns from. Click on the CODE, DATE, and QTY columns while holding down the **Ctrl** key.

- Click the left mouse button again and, keeping it held down, drag the selected columns to the output link in the upper-right pane. Drop the columns by releasing the mouse button. The columns appear in the top pane and the associated meta data appears in the lower-right pane.

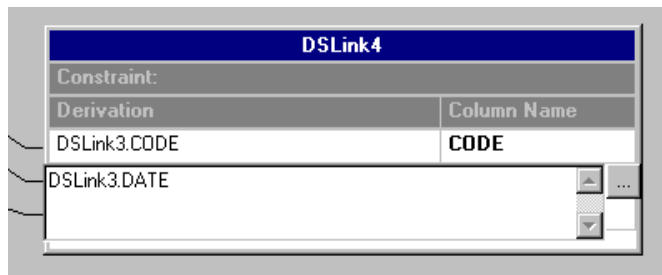


The next step is to edit the meta data for the output link. You will be transforming dates from *YYYY-MM-DD*, held in internal date format, to strings containing the date in the form *YYYY-MM*. You need to select a new SQL type and a new date element for the output DATE column, to specify that it will be carrying a string. You do this in the lower-right pane of the Transformer Editor.

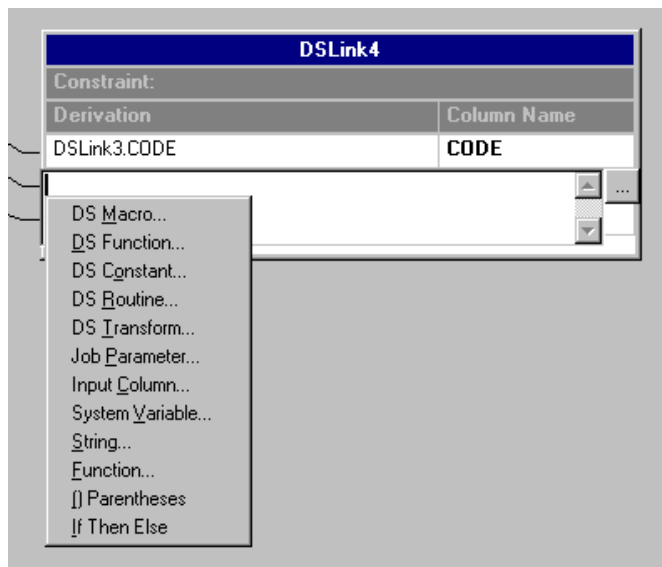
- In the **SQL type** field for the DSLink4 DATE column, select **Char** from the drop-down list.
- In the **Data element** field for the DSLink4 DATE column, select **MONTH.TAG** from the drop-down list.

Next you will specify the transform to apply to the input DATE column to produce the output DATE column. You do this in the upper-right pane of the Transformer Editor.

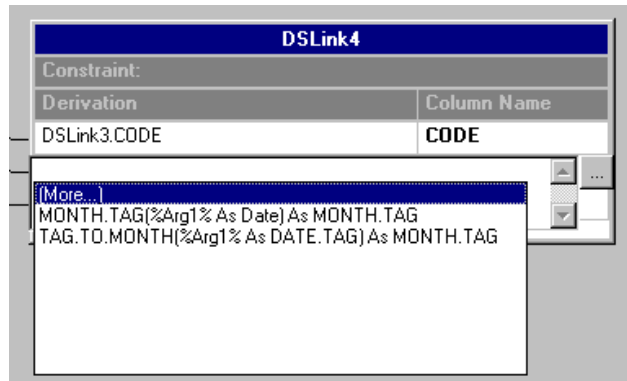
5. Double-click the **Derivation** field for the DSLink4 DATE column. The Expression Editor box appears. At the moment, the box contains the text `DSLInk3`, which indicates that the output DATE column is directly derived from the input DATE column. Select the text **DSLInk3** and delete it by pressing the **Delete** key.



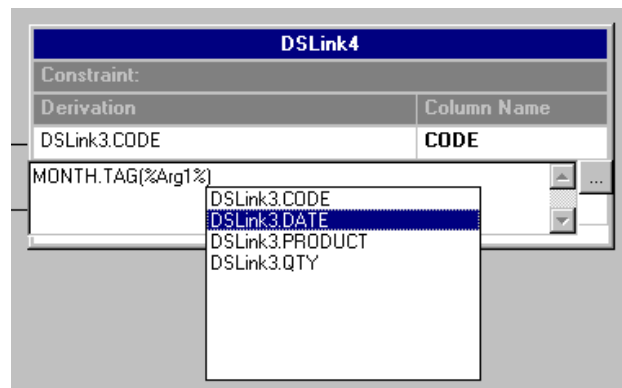
6. Right-click in the Expression Editor box to invoke the Suggest Operand menu.



7. Select **DS Transform...** from the menu. The Expression Editor then displays the transforms that are applicable to the MONTH.TAG data element:



8. Select the **MONTH.TAG** transform. It appears in the Expression Editor box with the argument field [%Arg1%] highlighted.
9. Right-click to invoke the Suggest Operand menu again. This time, select **Input Column**. A list of available input columns appears.



10. Select **DSLink3.DATE**. This then becomes the argument for the transform.
11. Click **OK** to save the changes and exit the Transformer Editor.

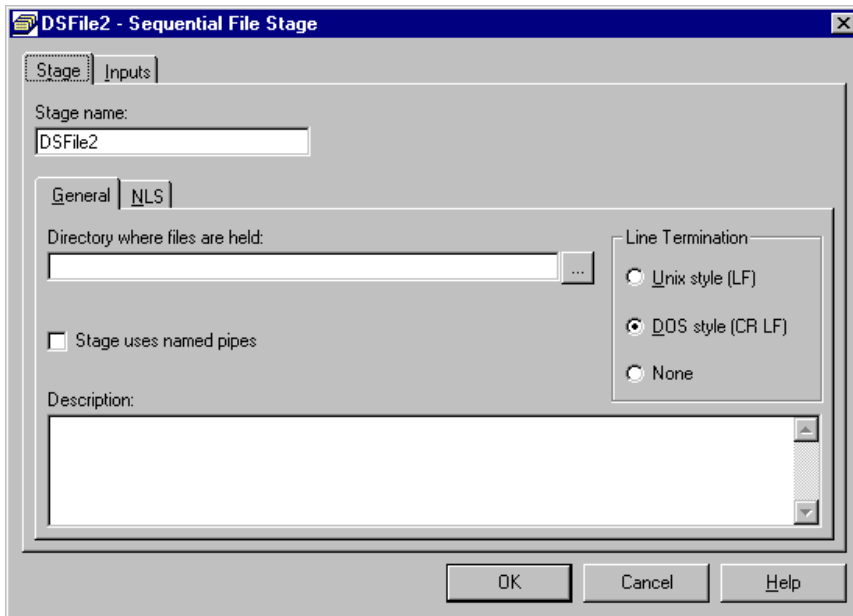
### **Advanced Procedures**

To create and use custom transforms, see “Transforms” on page 11-15.

## Editing the Sequential File Stage

The data warehouse is represented by a Sequential File stage. The data to be written to the data warehouse is already specified in the Transformer stage. However, you must enter the name of a file to which the data is written when the job runs. If the file does not exist, it is created.

Double-click the stage to edit it. The Sequential File Stage dialog box appears:



This dialog box has two pages:

- **Stage.** Displayed by default. This page contains the name of the stage you are editing and two further pages. The **General** page specifies the directory path of the sequential file, and the **NLS** page specifies a character set map to use with the stage.
- **Inputs.** Describes the data flowing into the stage. This page only appears when you have an input to a Sequential File stage. You do not need to edit the column definitions on this page, because they were all specified in the Transformer stage.



To edit the Sequential File stage:

1. Enter the directory path where the new file will be created in the **Directory where files are held** field. By default, this field is empty. If you leave it empty, the default installation directory is used. If you want to choose an alternative directory, click **Browse...** to search the DataStage server for a suitable directory path.
2. Click the **Inputs** tab. The **Inputs** page appears. This page contains:
  - The name of the link. This is automatically set to the link name used in the job design.
  - **General** page. Contains the name of the file, an optional description of the link, and update action choices. You can use the default settings for this example, but you must enter a file name.
  - **Format** page. Determines how the data is written to the file. In this example, the data is written using the default settings, that is, as a comma-delimited file.
  - **Columns** page. Contains the column definitions for the data you want to extract. This page contains the column definitions specified in the Transformer stage's output link.
3. Enter the name of the text file you want to create in the **File name** field, for example, **MyFile.txt**.
4. Click **OK** to close the Sequential File Stage dialog box.
5. Choose **File > Save Job** to save the job design.

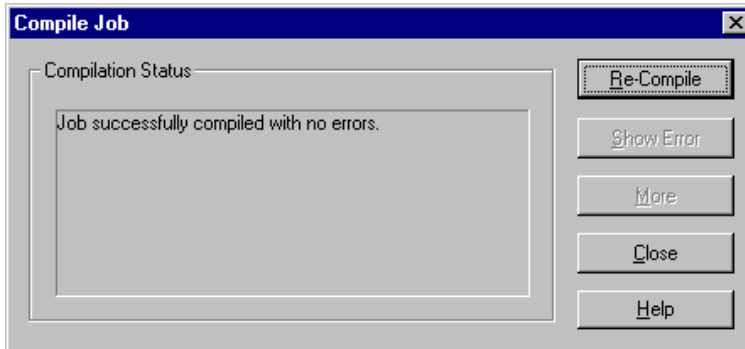
The job design is now complete and ready to be compiled.

## Compiling a Job

When you finish your design you must compile it to create an executable job. Jobs are compiled using the Designer. To compile the job, do one of the following:

- Choose **File > Compile**.
- Click the **Compile** icon on the toolbar.

The Compile Job window appears:



The job is compiled. The result of the compilation appears in the display area. If the result of the compilation is `Job successfully compiled with no errors` you can go on to schedule or run the job. The executable version of the job is stored in your project along with your job design.

If an error is displayed, click **Show Error**. The stage where the problem occurs is highlighted in the job design. Check that all the input and output column definitions have been specified correctly, and that you have entered directory paths and file or table names where appropriate.

For more information about the error, click **More**. Click **Close** to close the Compile Job window.

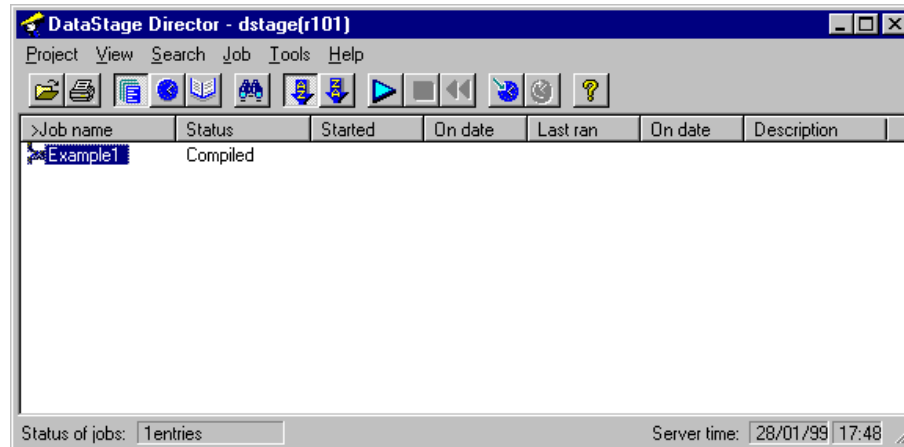
### ***Advanced Procedures***

For more information about compiling and releasing a job, see Chapter 17.

## **Running a Job**

Executable jobs are scheduled by the DataStage Director and run by the DataStage Server. You can start the Director from the Designer by choosing **Tools ► Run Director**.

When the Director is started, the DataStage Director window appears with the status of all the jobs in your project:



Highlight your job in the **Job name** column. To run the job, choose **Job ► Run Now** or click the **Run** icon on the toolbar. The Job Run Options dialog box appears and allows you to specify any parameter values and to specify any job run limits. Supply the required information, then click **Run**. The status changes to Running. When the job is complete, the status changes to Finished.

Choose **File ► Exit** to close the DataStage Director window.

Refer to *DataStage Operator's Guide* for more information about scheduling and running jobs.

### **Advanced Procedures**

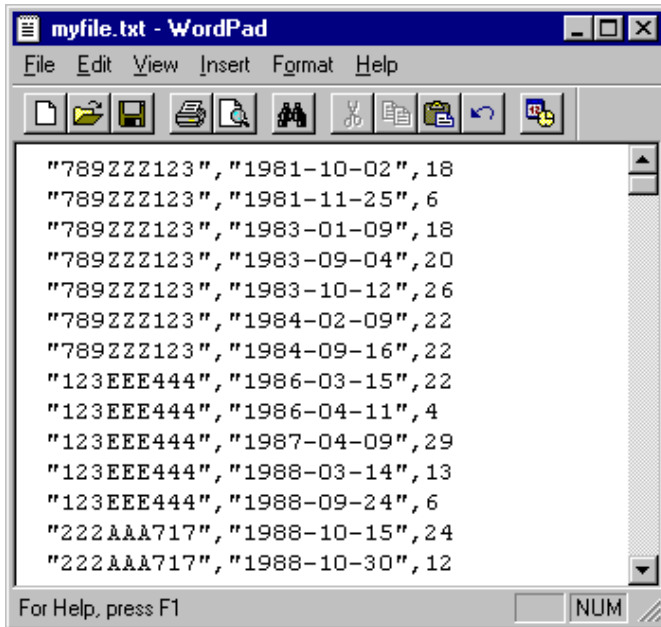
It is possible to run a job from within another job. For more information, see "Job Control Routines" on page 5-38.

## **Analyzing Your Data Warehouse**

When you have data in your data mart or data warehouse, you can use any BI (business intelligence) tool (for example, Cognos Impromptu or PowerPlay) to analyze and report on the data.

In the example, you can confirm that the data was converted and loaded correctly by viewing your text file using the Windows WordPad. Alternatively, you could use the built-in Data Browser to view the data from the **Outputs** page of the

Sequential File stage. See “Using the Data Browser” on page 5-24. The following is an example of the file produced in WordPad:



# 4

## Setting Up a Project

This chapter describes how to set up your project using the DataStage Manager before you start to develop DataStage jobs.

**Note:** Before users can develop or run DataStage applications, you must define who has access to the DataStage projects. This configuration is done using the DataStage Administrator. For more information, see *DataStage Administrator's Guide*.

Before you develop any DataStage job, you must plan and set up your project. This includes:

- Assessing your data
- Creating the data warehouse

You can then use the DataStage Manager to:

- Import, create, and edit table definitions
- Import, create, and edit stored procedure definitions
- Create, edit, and assign data elements

### Assessing Your Data

Before you design your application you must assess your data. DataStage jobs can be complex and so it is advisable to consider the following before starting a job:

- The number and type of data sources. You need a stage for each data source you want to access. For each different type of data source you need a different type of stage.
- The location of the data. Is your data on a networked disk or a tape? You may find that if your data is on a tape, you need to arrange for a plug-in stage to extract the data.

- The content of the data. What columns are in your data? Can you import the table definitions, or do you need to define them manually? Are the definitions of the data items consistent between different data sources?
- The data warehouse. What do you want to store in the data warehouse and how do you want to store it?

## Creating the Data Warehouse

As well as assessing the data in your data sources, you must also determine the data you want to load into your data mart or data warehouse. The table definitions used to describe the tables you want to populate are used in the final stage of your DataStage job.

**Note:** The data warehouse must be configured before you can execute a DataStage job.

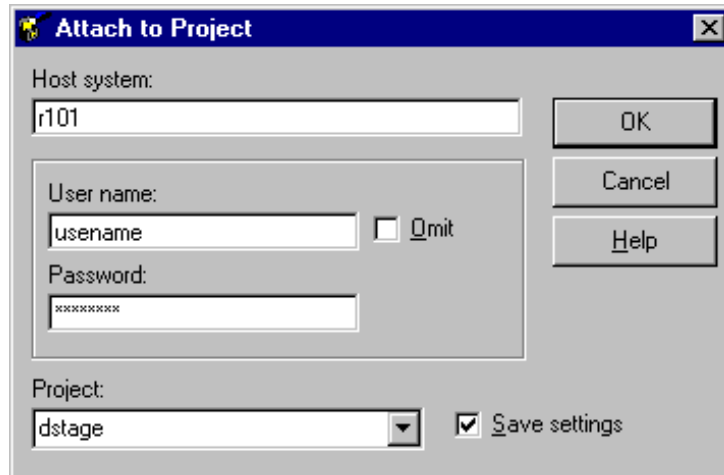
## The DataStage Manager

The DataStage Manager is used to:

- View and edit the contents of the Repository
- Import table or stored procedure definitions
- Create table or stored procedure definitions manually (if necessary)
- Create data elements
- Create routines, plug-ins, and custom transforms
- Assign data elements

## Starting the DataStage Manager

To start the DataStage Manager, choose **Start** ► **Programs** ► **Ardent DataStage** ► **DataStage Manager**. The Attach to Project dialog box appears:



To connect to a project:

1. Enter the name of your host in the **Host system** field. This is the name of the system where the DataStage Server components are installed.
2. Enter your user name in the **User name** field. This is your user name on the server system.
3. Enter your password in the **Password** field.

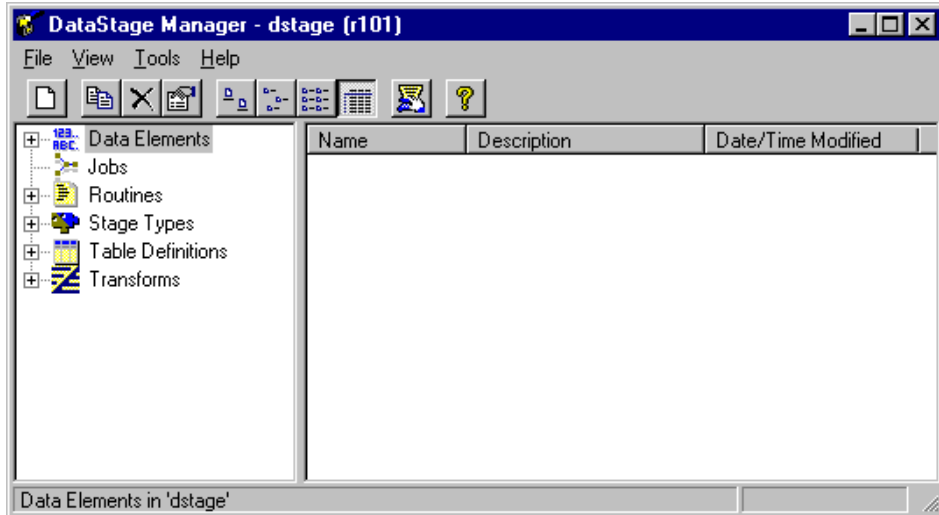
**Note:** If you are connecting to the server via LAN Manager, you can select the **Omit** check box. The **User name** and **Password** fields gray out and you log on to the server using your Windows NT Domain account details.

4. Choose the project to connect to from the Project drop-down list box. This list box displays the projects installed on your DataStage server.
5. Select the **Save settings** check box to save your logon settings.
6. Click **OK**. The DataStage Manager window appears.

**Note:** You can also start the DataStage Manager from the DataStage Designer or Director by choosing **Tools ► Run Manager**.

## The DataStage Manager Window

The DataStage Manager window appears after you have entered your project details:



The DataStage Manager window has the following components.

### Title Bar

The title bar displays the name of the project you are working in, followed by the name of the host where the DataStage Server components are installed. The title bar is updated if you choose to view another project in the Repository.

### Menu Bar

There are four pull-down menus:

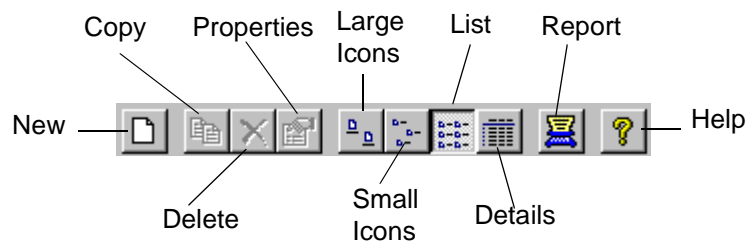
- **File.** Creates data elements, table definitions, stage types, and transforms. Sets up a default printer and copies, renames, and deletes components in the project tree.
- **View.** Specifies how information is displayed in the display area. Items can be displayed as large icons, small icons, a list, or with details. There are also options to refresh the display and to show or hide the toolbar and status bar.



- **Tools.** Imports and exports projects or job components. Also imports table definitions from data sources or data marts. Gives access to DataStage reporting facilities. Starts the DataStage Designer or Director and lets you invoke third-party applications, or add third-party applications to the Manager. This menu also contains options to release and package jobs and job components and to import or export meta data using MetaBrokers.
- **Help.** Invokes the Help system.

### Toolbar

The Manager toolbar contains the following icons:



### Project Tree

The project tree is in the left pane of the DataStage Manager window and contains a summary of the project contents. The tree is divided into six main branches:

- **Data Elements.** A branch exists for the built-in data elements and the additional ones you define.
- **Jobs.** A leaf exists under this branch for each job in the project.
- **Routines.** Each routine you create or import is stored under this branch.
- **Stage Types.** Each plug-in you create or import is stored under this branch.
- **Table Definitions.** Table definitions are stored according to the data source. If you import a table definition, a branch is created under the data source type (ODBC, UniVerse, Hashed, UniData, Sequential, or StoredProcedure). If you manually enter a table definition, you can create a new branch anywhere under the main **Table Definitions** branch.
- **Transforms.** A branch exists for the built-in transforms and for each group of custom transforms created.

Double-click a branch to expand it to a leaf (item) level. You can display the properties (definition) of an item by double-clicking it in the display area.

## Display Area

The display area is in the right pane of the DataStage Manager window and displays the contents of a chosen branch. You can display items in the display area in four ways:

- **Large icons.** Items are displayed as large icons arranged across the display area.
- **Small icons.** Items are displayed as small icons in the display area.
- **List.** Items are displayed in a list going down the display area.
- **Details.** Items are displayed in a table. The following information is displayed for each item:
  - **Name.** The name of the item.
  - **Description.** The description of the item.
  - **Date/Time Modified.** The date and time the branch or item was last modified.

If you double-click an item in the display area, a dialog box appears with the properties of the chosen item.

## Shortcut Menus

There are a number of shortcut menus available which you display by clicking the right mouse button. There are three types of menu:

- **Branch level.** Appears when you click the display area (not an item) or the project tree. Use this menu to refresh the display or to create a new item under the chosen branch.
- **Item level.** Appears when you click a highlighted item in the display area. Use this menu to copy, rename, delete, or display the properties of the chosen item.
- **Grid level.** Appears when you are editing a grid. Use this menu to edit, insert, or delete a row.

## Using the DataStage Manager

The DataStage Manager provides a means of viewing and managing the contents of the Repository. You can use the DataStage Manager to:

- Create items
- Rename items
- View and edit item properties
- Delete items
- Copy items

### Creating Items in the Repository

You can create new data elements, table definitions, or transforms. To create a new item, select the top-level branch in the project tree and do one of the following:

- Choose **File** ► **New**. The **New** option describes the selected item, for example, **File** ► **New Transform...**
- Choose **New...** from the shortcut menu.
- Click the **New** icon on the toolbar.

A dialog box appears for you to enter the properties of the new item. Click **OK** to save the new item. A leaf is created for the item in the project tree.

### Renaming Items in the Repository

You can rename any of the items in the project tree, except for the top-level branches, jobs, and built-in items.

To rename an item, select it in the display area and do one of the following:

- Click the item again. An edit box appears and you can enter a different name or edit the existing one. Save the new name by pressing **Enter** or by clicking outside the edit box.
- Double-click the item. A properties dialog box appears and you can edit the field containing the item's name. Click **OK** to save the new name.
- Choose **Rename** from the shortcut menu. An edit box appears and you can enter a different name or edit the existing one. Save the new name by pressing **Enter** or by clicking outside the edit box.

### Viewing or Editing Items in the Repository

You can view or edit the properties of any item in the project tree.

**Note:** You cannot edit the properties of built-in items.

To view or edit the properties of an item, select the item in the display area and do one of the following:

- Choose **File ► Properties...**
- Choose **Properties...** from the shortcut menu.
- Double-click the item in the display area.
- Click the **Properties** icon on the toolbar.

A dialog box appears with the item's properties. The content of the dialog box depends on the type of item you are viewing or editing. Click **OK** to save any changes and to close the dialog box.

### Deleting Items in the Repository

You can delete any item in the project tree, except the built-in items.

**CAUTION:** You must be absolutely certain you want to remove an item in the Repository before you delete it. If it is used by another item in the project, your jobs will fail to compile.

To delete an item, select it in the display area and do one of the following:

- Choose **File ► Delete**.
- Click the **Delete** icon on the toolbar.

A message box appears. Click **Yes** to delete the item.

### Copying Items in the Repository

You can copy items in the same branch in the project tree. To copy an item, select the item in the display area and do one of the following:

- Choose **File ► Copy**.
- Choose **Copy** from the shortcut menu.
- Click the **Copy** icon on the toolbar.

The item is copied and a new item is created under the same branch in the project tree. By default, the name of the copy is called CopyOfXXX, where XXX is the name of the chosen item. An edit box appears allowing you to rename the copy immediately.

### Choosing an Alternative Project

When you start the DataStage Manager, the project specified in the Attach to Project dialog box is opened. You can choose to view any DataStage project in the DataStage Manager.

To choose an alternative project:

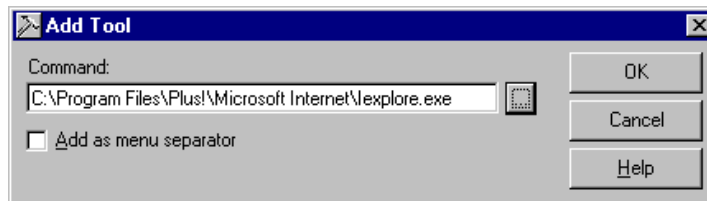
1. Choose **File ► Open Project...** . The Open Project dialog box appears.
2. Choose the project you want to open from the Projects list box. This list box contains all the DataStage projects on the DataStage server (specified in the **Host system** field).
3. Click **OK** to open the chosen project. The DataStage Manager window is updated and displays the contents of the new project.

To open a project on a different DataStage server, click **New host...** . The Attach to Project dialog box appears. You must enter the name of the DataStage server and your logon details before choosing the project from the Project drop-down list box. Click **OK** to open the chosen project. The DataStage Manager window is updated and displays the contents of the new project.

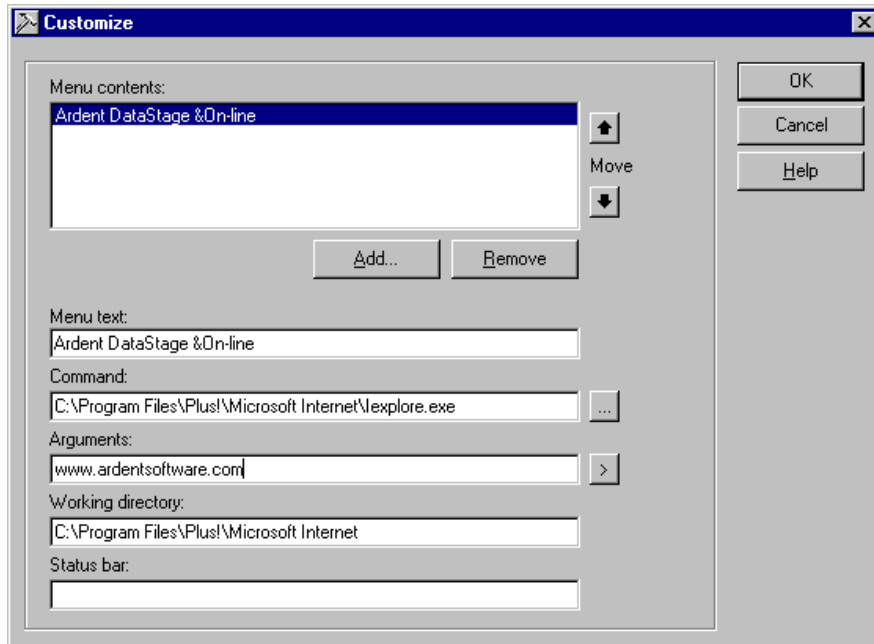
## Customizing the Tools Menu

You can configure the DataStage Manager so that you can invoke third-party applications from the Tools menu. To configure the menu:

1. Choose **Tools ► Custom ► Customize...** . The Customize dialog box appears.
2. Click the **Add...** button to display the Add Tool dialog box.



3. Type in, or browse for, the executable of the application that you are adding and click **OK** to return to the Customize dialog box. The application appears in the Menu contents list box.



4. Type in the menu name for the application in the **Menu text** field, and specify its order in the menu using the arrow keys and the **Menu contents** field.
5. Specify any required command line arguments for the tool in the **Arguments** field. Alternatively click the > button and choose from a predefined list of argument tokens.
6. Click **OK** to add the application to the Manager's Tools menu (**Tools > Custom**).

**Note:** Any applications that you add to the DataStage Manager's Tools menu are added automatically to the Tools menu of other DataStage applications (e.g., Designer and Director).

## Exiting the DataStage Manager

To exit the DataStage Manager, choose **File > Exit** from the DataStage Manager window.

## Table Definitions

Table definitions are the key to your DataStage project and specify the data to be used at each stage of a DataStage job. Table definitions are stored in the Repository and are shared by all the jobs in a project. You need, as a minimum, table definitions for each data source and one for each data target in the data mart.

You can import, create, or edit a table definition using the DataStage Manager.

You can use the Data Browser to view the actual data in data sources from which you are importing table definitions.

### Importing a Table Definition

The easiest way to specify a table definition is to import it directly from the source or target database. A new table definition is created and the properties are automatically filled in with the details of your data source or data mart.

You can import table definitions from an ODBC data source, certain plug-in stages (including Sybase Open Client and Oracle OCI), a UniVerse table, a hashed (UniVerse) file, a UniData file, or a sequential file.

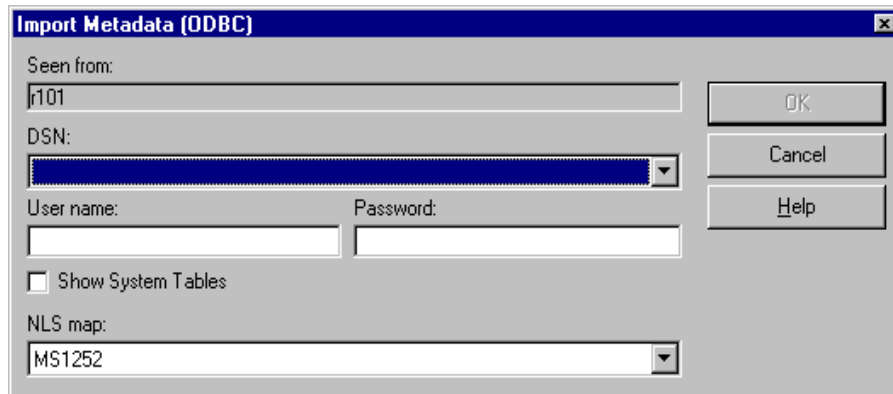
You can also import table definitions from other data warehousing tools via a MetaBroker. For more information about MetaBrokers, see Chapter 20, “Using MetaBrokers.”

### Importing from an ODBC Data Source

To import a table definition from a database via an ODBC connection:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.

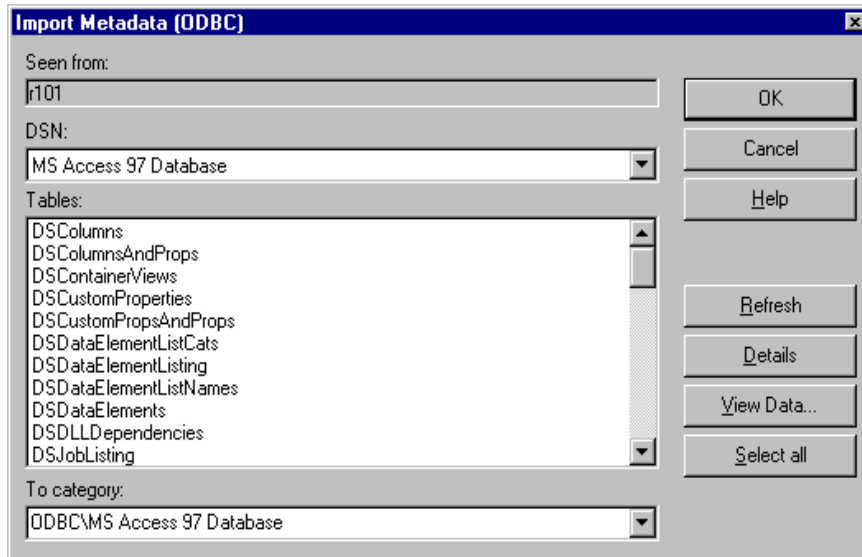
2. Choose **Tools** ► **Import** ► **ODBC Table Definitions...** . The Import Metadata (ODBC) dialog box appears:



3. Choose a data source name from the DSN drop-down list box. This list box contains the ODBC data sources defined on the server. For more information about defining ODBC data sources, see *DataStage Administrator's Guide*.
4. Enter your user name in the **User name** field.
5. Enter your password in the **Password** field.  
**Note:** The user name and password are those required to access the system specified by the DSN and might not be the same as those used to access DataStage. Also note that the system specified by the DSN might not be the same as the one named in the **Seen From** field.
6. If NLS is enabled, choose a character set map from the NLS map drop-down list. This map defines the character set that the ODBC data uses.
7. Select the **Show System Tables** check box if you want system tables to be included in the list of available tables.



- Click **OK**. A connection is made to the ODBC data source and the updated Import Metadata (ODBC) dialog box displays the available tables. The **To category** field also displays the branch in the Repository where the table definition will be stored.



- Select a table or tables from the Tables list box or click **Select all** to choose all available tables.

You can click **Details** to display further information about each table or **View Data...** to invoke the Data Browser to examine the actual data in the table. You can also edit the **To category** field or choose a new category from the drop-down list box. The entry here determines where the table definition will be stored under the **Table Definitions** branch in the Repository.

Click **Refresh** to update the list of tables if you think the database contents have changed since the list was created.

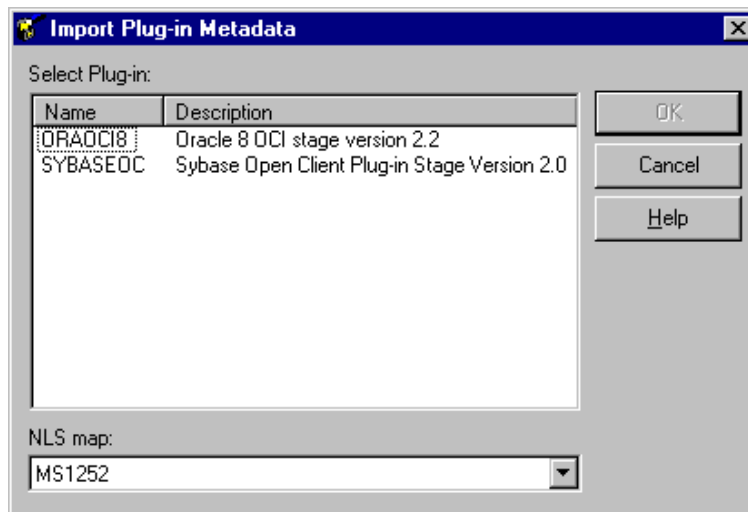
- Click **OK**. The table definition for the chosen table or tables is imported and a new branch is created under the **Table Definitions** branch in the project tree.

### Importing from a Plug-In

To import a table definition from a registered plug-in:

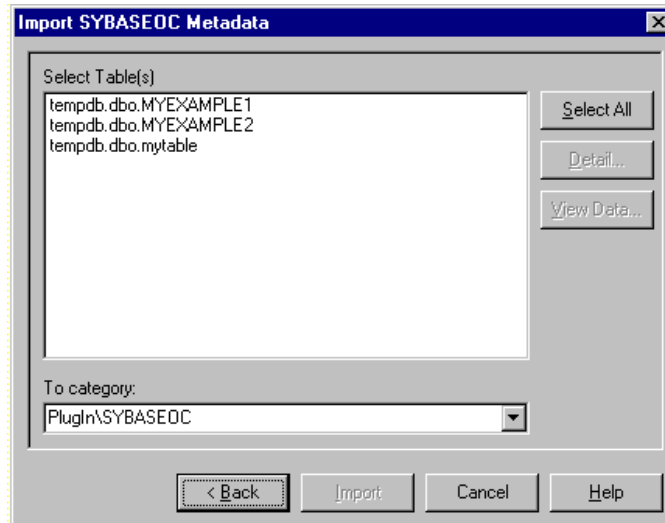
- Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.

2. Choose **Tools** ► **Import** ► **Plug-in Metadata Definitions...** . The Import Plug-in Metadata dialog box appears, listing the plug-ins for which import meta data facilities are available:



3. Choose the particular plug-in that you want to import meta data from, then click **OK**. This invokes a wizard-style dialog that guides you through the rest of the import process.
4. The wizard prompts you for information needed by the plug-in to generate a list of available tables. Click **Next** at each stage. This cycle continues until all the necessary information has been collected.
5. The wizard displays a list of meta data items available for import. The wizard screen for the Sybase Open Client (SYBASEOC) plug-in is shown below.

When you have selected all the items to import and specified the destination category, click **Import**.

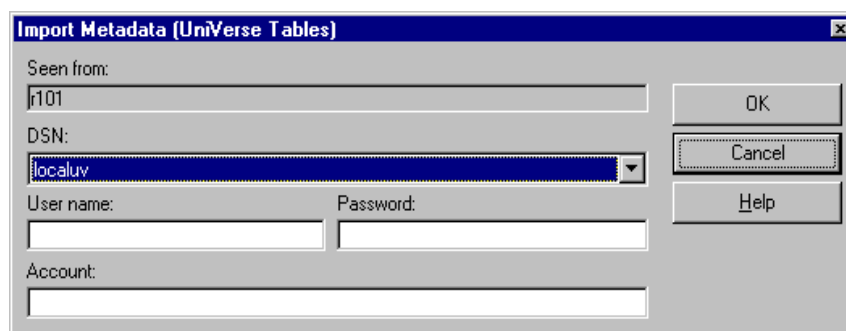


The table definition for the chosen meta data is imported and a new branch is created under the **Table Definitions** branch in the project tree.

### Importing from a UniVerse Table

To import a table definition from a UniVerse table:

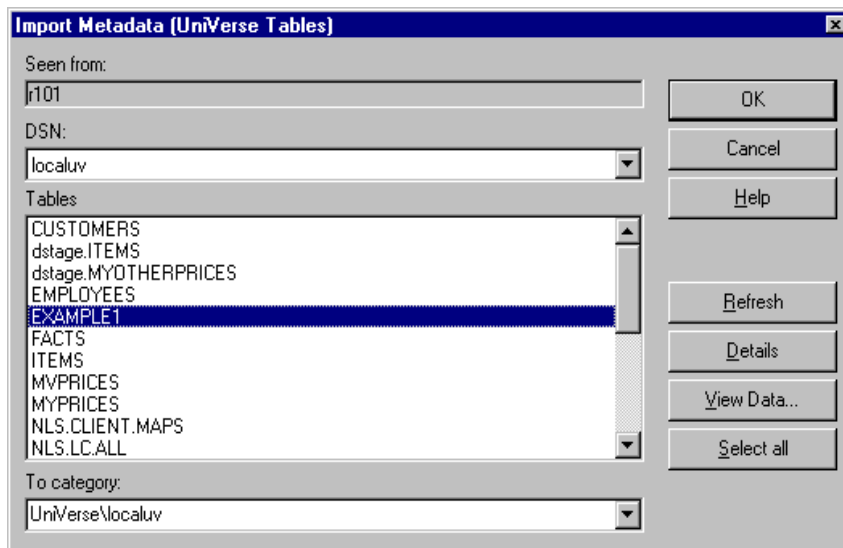
1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools** ► **Import** ► **UniVerse Table Definitions...** . The Import Meta-data (UniVerse Tables) dialog box appears:



3. Choose a data source name from the **DSN** drop-down list box. This list box contains the local and remote UniVerse data sources defined in the *uvodbc.config* file. For more information, see *DataStage Administrator's Guide*.
4. Enter your user name in the **User name** field.
5. Enter your password in the **Password** field.
 

**Note:** The user name and password are those required to access the system specified by the DSN and might not be the same as those used to access DataStage. Also note that the system specified by the DSN might not be the same as the one named in the **Seen From** field.
6. Enter an account name in the **Account** field.
 

**Note:** When you choose **localuv** as the data source name, the **User name**, **Password**, and **Account** fields gray out.
7. If NLS is enabled and you are connecting to a remote UniVerse source, choose a character set map from the **NLS map** drop-down list. This map defines the character set that the UniVerse table uses.
8. Click **OK**. A connection is made to the UniVerse data source and the updated Import Metadata (UniVerse Tables) dialog box displays the available tables. The **To category** field also displays the branch in the Repository where the table definition will be stored.



9. Select a table or tables from the **Tables** list box or click **Select all** to choose all available tables.

You can click **Details** to display further information about each table or **View Data...** to invoke the Data Browser to examine the actual data in the table. You can also edit the **To category** field or choose a new category from the drop-down list box. The entry here determines where the table definition will be stored under the **Table Definitions** branch in the Repository.

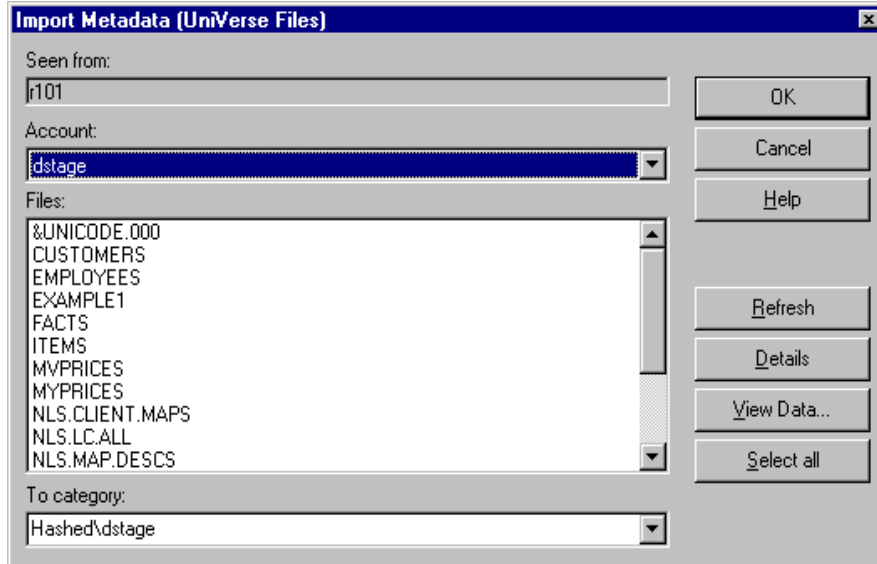
Click **Refresh** to update the tables in the Tables list box.

10. Click **OK**. The table definition for the chosen table or tables is imported and a new branch is created under the **Table Definitions** branch in the project tree.

### Importing from a Hashed File

To import a table definition from a hashed (local UniVerse) file:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools** ► **Import** ► **UniVerse File Definitions...** . The Import Metadata (UniVerse Files) dialog box appears:



3. Choose an account from the Account drop-down list box. This list box contains all the accounts in the UV.ACCOUNT file (except uv) for the local UniVerse system. The updated Files list box displays the files in the chosen account.
4. Select a file or files from the Files list box or click **Select all** to highlight all the files. You can click **Details** to display further information about each file or **View Data...** to invoke the Data Browser to examine the actual data in the file. Click **Refresh** to update the files in the Files list box.
5. Enter a category in the **To category** field or choose one from the drop-down list box. The entry here determines where the table definition will be stored under the main **Table Definitions** branch in the Repository.
6. Click **OK**. The table definition for the chosen file or files is imported and a new branch is created under the **Table Definitions** branch in the project tree.

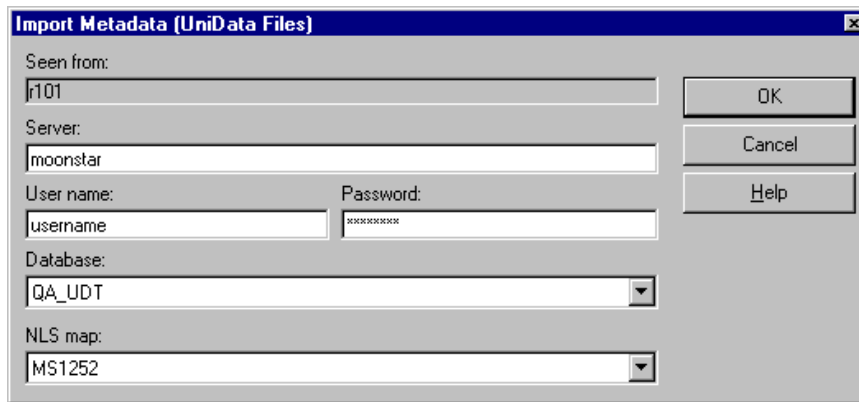
You can edit the table definition to remove unwanted column definitions, assign data elements, or change branch names.

**Note:** UniVerse files are the only supported hashed files.

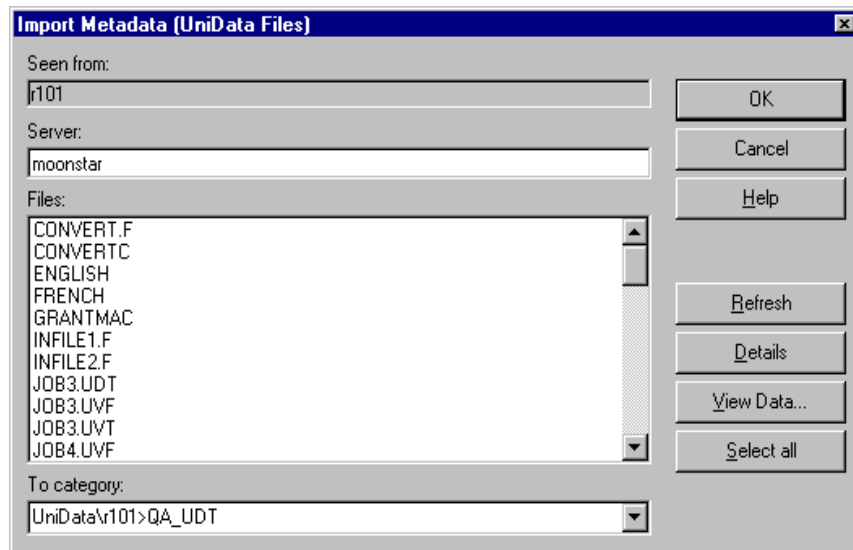
### Importing from a UniData File

To import a table definition from a UniData file:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools** ► **Import** ► **UniData File Definitions...** . The Import Metadata (Unidata Files) dialog box appears:



3. Enter the name of the host where the UniData files are located in the **Server** field.
4. Enter your user name in the **User name** field.
5. Enter your password in the **Password** field.
6. If NLS is enabled, choose a character set map from the NLS map drop-down list. This map defines the character set that the UniData file uses.
7. Click the Database drop-down list box to populate it with the names of UniData databases on that server.
8. Choose a database and click **OK**. The updated Import Metadata (Unidata Files) dialog box displays a list of available UniData files and tables:



9. Choose the files or tables that you want to import meta data from and click **OK**. The table definitions for the chosen files or tables are imported and, if such a branch does not already exist, a new branch is created under the **Table Definitions** branch in the project tree. Or you can choose a branch from the **To category** list box. Categories are in the form `Unidata\Server>Database`.

If you encounter an error when connecting to a UniData data source, such as:

```
UniData Client error: call to UniOpenPos returned 45 - Client
version (11) and server version (12) are incompatible
```

you need to edit the *UNIAPI.INI* file in the Windows directory on your client machine to change the value of the *PROTOCOL* variable to match the server version. So, in the case of the example error, you would need to change the variable value from 11 to 12:

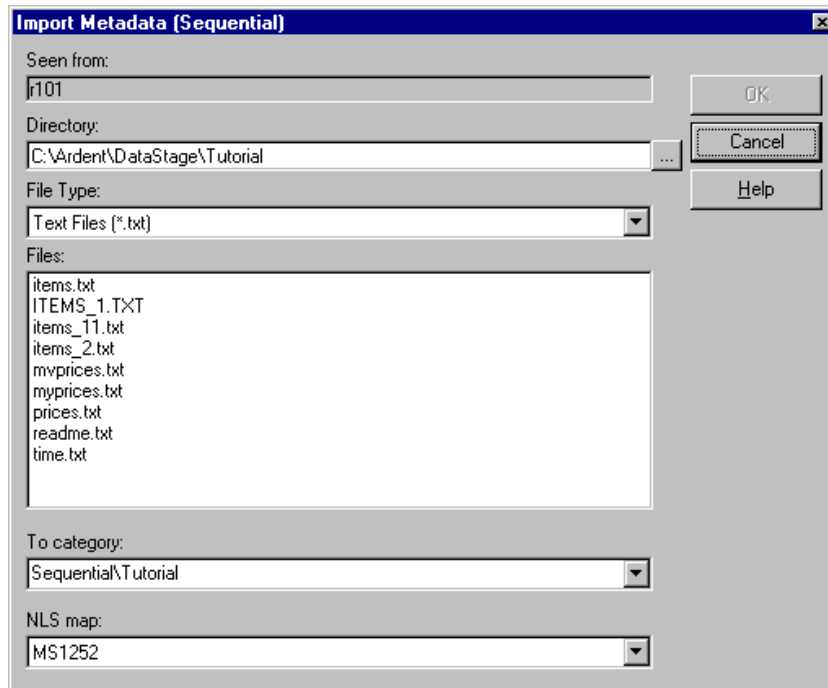
```
PROTOCOL = 12
```

## Importing from a Sequential File

You can import meta data from a sequential file that has fixed-width fields or separated values. When you import meta data from a sequential file, you must specify the name of the file and its location, plus a definition of the file contents. This definition is used to import the meta data.

To import a table definition from a sequential file:

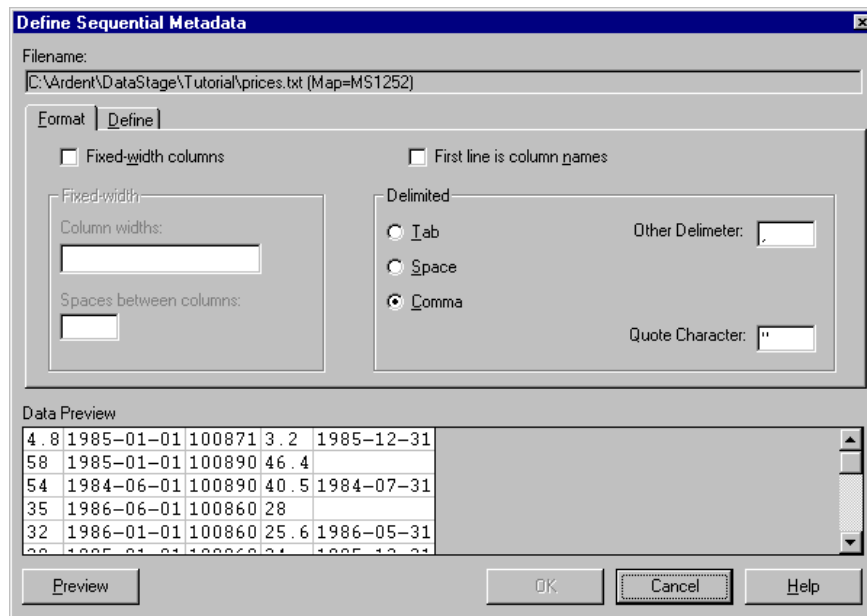
1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools** ► **Import** ► **Sequential File Definitions...** . The Import Metadata (Sequential) dialog box appears:





3. Specify the directory path where the file is found in the **Directory** field. You can enter a path directly or click ... to search the system for a suitable directory.
4. Choose the type of file from the File Type drop-down list box:
  - **Text Files (\*.txt)**. This is the default setting.
  - **Comma Separated (\*.csv)**
  - **All Files (\*.\*)**

The updated Files list box displays all the files of the chosen type in the specified directory.
5. Choose the file you want from the Files list box.
6. Enter a category in the **To category** field or choose one from the drop-down list box. The entry here determines where the table definition will be stored under the main **Table Definitions** branch in the Repository.
7. If NLS is enabled, choose a character set map from the NLS map drop-down list. This map defines the character set that the sequential file uses.
8. Click **OK**. The Define Sequential Metadata dialog box appears:



This dialog box has the following features:

- **Filename** field. Displays the name and path of the sequential file. You cannot edit this field. If you want to choose an alternative file, click **Cancel**. The Import Metadata (Sequential) dialog box reappears and you can specify a different directory or file.
- **Format** page. Displayed by default. Contains options that define how the file is formatted. There are two check boxes on the left of this page:
  - **Fixed-width columns**. Specifies whether the file contains fixed-width columns. The setting for this check box determines the options available on the rest of the page. When this check box is cleared (the default setting) the options in the Delimited area are available. Select this check box if the file contains fixed-width columns. You must then use the options in the Fixed width area to define the column widths.
  - **First line is column names**. Specifies that the first data row in the file contains the column names. This check box is cleared by default. Select this check box if the first line contains column names.

The rest of this page is divided into two areas:

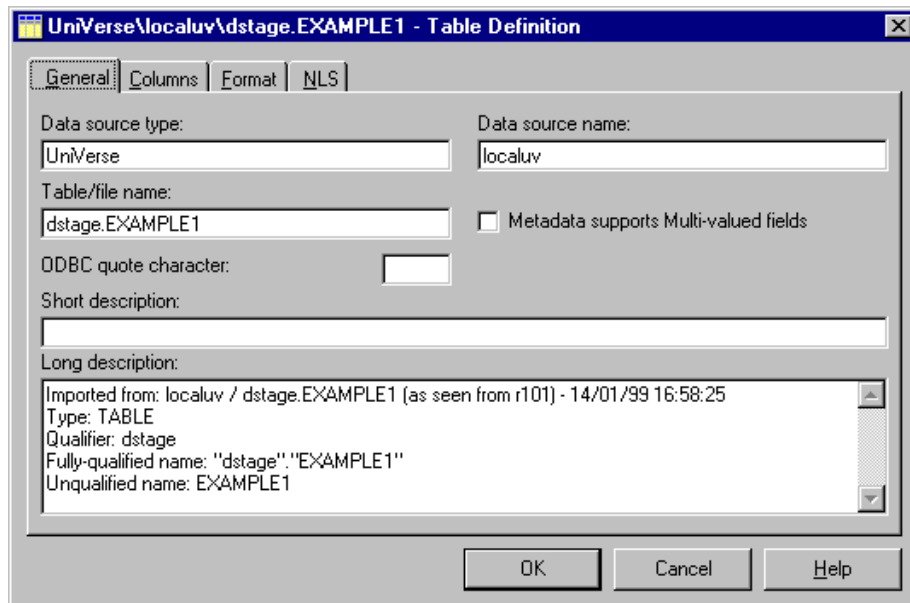
- **Delimited**. The options in this area are available only if the **Fixed-width columns** check box is cleared. The option buttons **Tab**, **Space**, and **Comma** (the default) specify how the fields are separated. To specify an alternative separation character, enter it in the **User** field. If values are quoted, enter the quote character in the **Quote Character** field. You can enter a number from 0 to 255 in the **User** or **Quote Character** fields to represent the ASCII code for the character you want to use.
- **Fixed width**. The options in this area are available only if the **Fixed-width columns** check box is selected. Enter the width for each column in the **Column widths** field. Widths are entered using the format  $n,n,n\dots n$ , where  $n$  is the width of the column. You must have an entry for each column in the file. Specify the space between the columns by entering a value in the **Spaces between columns** field.
- **Define** page. Displays the column definition for each column in the file. If the first row in the file contains the column names, these are displayed in the **Column name** column, otherwise, default column names are assigned. You can edit any of the values in the grid. For more information about editing grids, see Appendix B, “Editing Grids.”

- **Data Preview** grid. This grid displays the first 10 rows of data in the chosen file using the options specified on the **Format** page. Click **Preview** to update the Data Preview grid using the format options you selected.
9. Complete the relevant details on the **Format** page and fine tune the SQL properties on the **Define** page.
  10. Click **OK** to import the table definition. The table definition for the chosen file is imported and a new branch is created under the **Table Definitions** branch in the project tree.

**Note:** The **OK** button is available only when you have viewed the **Define** page and checked the column definitions.

## The Table Definition Dialog Box

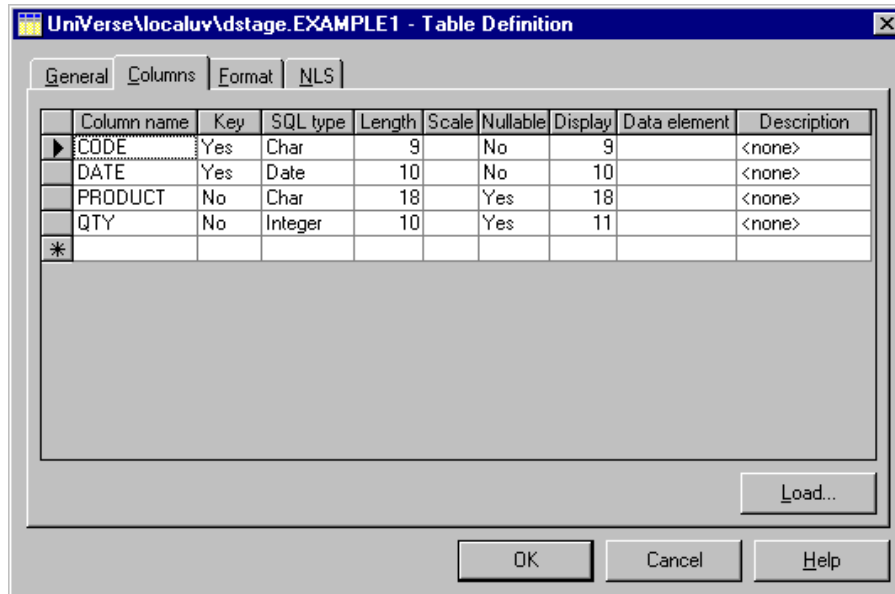
When you create, edit, or view a table definition using the DataStage Manager, the Table Definition dialog box appears:



This dialog box has up to four pages:

- **General.** This page contains general information about the table definition. The following fields are on this page:
  - **Data source type.** The type of data source, for example, UniVerse.
  - **Data source name.** If you imported the table definition, this contains a reference to where the original data is found. For UniVerse and ODBC data sources, this is the data source name. For hashed file data sources, this is an account name. For sequential file sources, this is the last component of the directory path where the sequential file is found.
  - **Table/file name.** The table or file name containing the data.
  - **Metadata supports Multi-valued fields.** Select this check box if the meta data supports multivalued data. If the check box is selected, three extra grid columns used for multivalued data support will appear on the **Columns** page. The check box is disabled for ODBC and stored procedure table definitions.
  - **ODBC quote character.** Allows you to specify what character an ODBC data source uses as a quote character.
  - **Short description.** A brief description of the data.
  - **Long description.** A long description of the data.

The combination of the data source type, data source name, and table or file name forms a unique identifier for the table definition. No two table definitions can have the same identifier.



- **Columns.** This page contains a grid displaying the column definitions for each column in the table definition. The grid has the following columns:
  - **Column name.** The name of the column.
  - **Key.** Indicates whether the column is part of the primary key.
  - **SQL type.** The SQL data type.
  - **Length.** The data precision. This is the length for CHAR data and the maximum length for VARCHAR data.
  - **Scale.** The data scale factor.
  - **Nullable.** Specifies whether the column can contain null values.
  - **Display.** The maximum number of characters required to display the column data.
  - **Data element.** The type of data in the column.
  - **Description.** A text description of the column.

The following columns appear if you selected the **Metadata supports Multi-valued fields** check box on the **General** page:

- **Association.** The name of the association (if any) that the column belongs to.

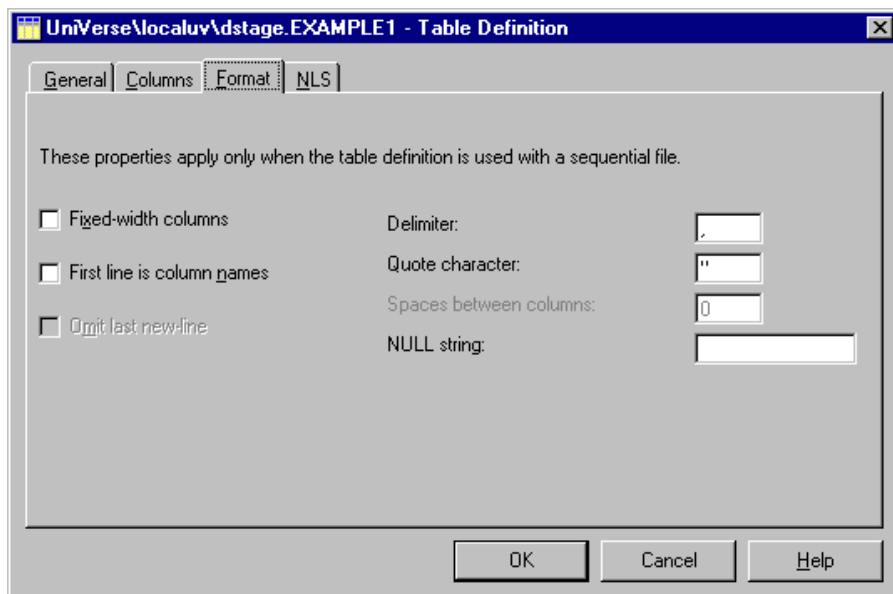
- **Position.** The field number.
- **Type.** The nesting type, which can be S, M, MV, or MS.

The following column may appear if NLS is enabled and the data source is Sequential, ODBC, or a generic plug-in:

- **NLS Map.** This property is visible only if NLS is enabled and **Allow per-column mapping** has been selected on the **NLS** page of the Table Definition dialog box. It allows you to specify a separate character set map for a column (which overrides the map set for the project or table).

The **Columns** page for each link also contains a **Load...** button. This loads (copies) the column definitions from a table definition elsewhere in the Repository.

A shortcut menu available in grids allows you to edit a cell, delete a row, or add a row. For more information about editing a grid, see Appendix B, "Editing Grids."



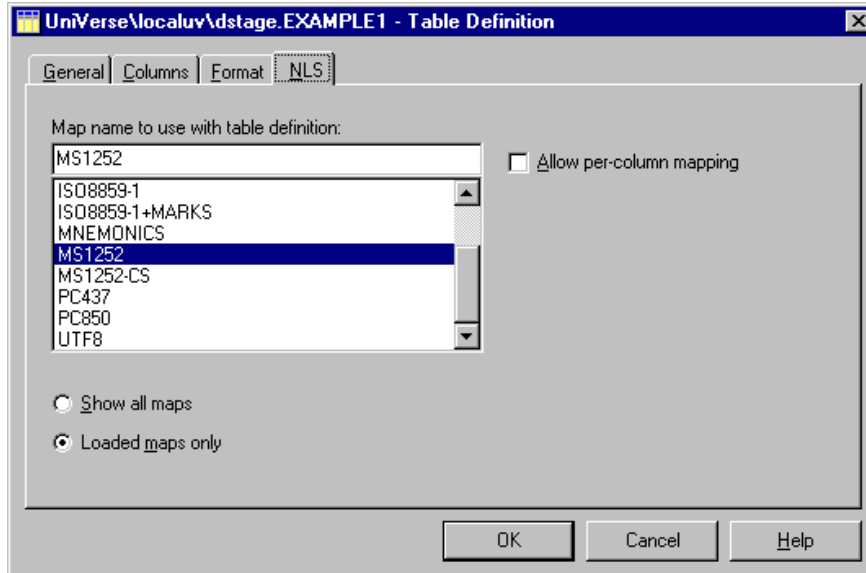
- **Format.** This page contains file format parameters for sequential files. These fields are automatically set when you import a table definition from a sequential file.

There are three check boxes on this page:

- **Fixed-width columns.** Specifies whether the sequential file contains fixed-width fields. This check box is cleared by default, that is, the file does not contain fixed-width fields. When this check box is selected, the **Spaces between columns** field appears.
- **First line is column names.** Specifies whether the first line in the file contains the column names. This check box is cleared by default, that is, the first row in the file does not contain the column names.
- **Omit last new-line.** Specifies whether the last newline character in the file is ignored. By default this check box is cleared, that is, if a newline character exists in the file, it is used.

The rest of this page contains four fields. The available fields depend on the settings for the check boxes.

- **Delimiter.** Contains the delimiter that separates the data fields in the file. By default this field contains a comma. You can enter a single printable character (other than a digit) or a number from 0 through 255 to represent the ASCII code for the character you want to use.
- **Quote character.** Contains the character used to enclose a data value that contains the delimiter character as data. By default this field contains a double quotation mark. You can enter a single printable character (other than a digit) or a number from 0 through 255 to represent the ASCII code for the character you want to use.
- **Spaces between columns.** Specifies the number of spaces used between the columns in the file. This field appears when you select **Fixed-width columns**.
- **NULL string.** Contains characters that are written to the file when a column contains SQL null values.



- **NLS.** If NLS is enabled, this page contains the name of the map to use for the table definitions. The map should match the character set used in the definitions. By default, the list box shows all the maps that are loaded and ready to use. **Show all maps** lists all the maps that are shipped with DataStage.

**Note:** You cannot use a map unless it is loaded into DataStage. You can load different maps using the DataStage Administrator. For more information, see *DataStage Administrator's Guide*.

Click **Allow per-column mapping** if you want to assign different character set maps to individual columns.

## Manually Entering a Table Definition

If you are unable to import the table definitions for your source or target data, you must enter this information manually.

To manually enter table definition properties, you must first create a new table definition. You can then enter suitable settings for the general properties before specifying the column definitions. You only need to specify file format settings for a sequential file table definition.



## Creating a Table Definition

To create a table definition:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Select the **Table Definitions** branch and do one of the following:
  - Choose **File ► New Table Definition...**
  - Choose **New Table Definition...** from the shortcut menu.
  - Click the **New** icon on the toolbar.

The Table Definition dialog box appears. You must enter suitable values in the fields on the **General** page.

3. Enter the type of data source in the **Data source type** field. The name entered here determines how the definition appears under the **Table Definitions** branch.
4. Enter the name of the data source in the **Data source name** field. This forms the second part of the table definition identifier and is the name of the branch created under the data source type branch.
5. Enter the name of the table or file containing the data in the **Table/file name** field. This is the last part of the table definition identifier and is the name of the leaf created under the data source branch.
6. Select the **Metadata supports Multi-valued fields** check box if the meta data supports multivalued data.
7. Enter a brief description of the data in the **Short description** field. This is an optional field. The text entered here is displayed when you choose **View ► Details** from the DataStage Manager window.
8. Enter a more detailed description of the data in the **Long description** field. This is an optional field.
9. Click the **Columns** tab. The **Columns** page appears at the front of the Table Definition dialog box. You can now enter or load column definitions for your data.

## Entering Column Definitions

You can enter column definitions directly in the Columns grid. To add a new column definition, you must edit the empty row (indicated by a star) in the Columns grid.

1. Click the first cell in the row (**Column name**) and enter the name of the column you want to add. This is the minimum requirement for adding a new column definition. You can then click the other cells you want to edit, or press **Tab** to move between the cells. For more information about adding and deleting rows, or moving between the cells of a grid, see Appendix B, “Editing Grids.”
2. Specify whether this column is a key column. If it is, choose **Y** from the drop-down list in the **Key** cell.
3. Specify the SQL data type by choosing an appropriate type from the drop-down list in the **SQL type** cell.
4. Enter an appropriate value for the data precision in the **Length** cell.
5. Enter an appropriate data scale factor in the **Scale** cell.
6. Specify whether the column can contain null values by choosing an appropriate option from the drop-down list in the **Nullable** cell.
7. Enter the maximum number of characters required to display the column data in the **Display** cell.
8. Choose the type of data the column contains from the drop-down list in the **Data element** cell. This list contains all the built-in data elements supplied with DataStage and any additional data elements you have defined. You do not need to edit this cell to create a column definition. You can assign a data element at any point during the development of your job. For more information, see “Assigning Data Elements” on page 4-46.
9. Enter text to describe the column in the **Description** cell. This cell expands to a drop-down text entry box if you enter more characters than the display width of the column. You can increase the display width of the column if you want to see the full text description. If you want to divide the text into paragraphs, press **Ctrl-Enter** at the point where you want to end each paragraph.
10. If you previously selected **Metadata supports Multi-valued fields**, then you will also need to fill in the **Association**, **Position**, and **Type** cells. Enter the name of the association the column belongs to (if any) in **Association**, enter the field number in **Position**, then choose **S**, **M**, **MV**, or **MS** from the Type drop-down list.
11. If you selected **Allow per-column mapping** on the **NLS** page, select the NLS map name that you want to use in the column from the drop-down list in the **NLS Map** cell.

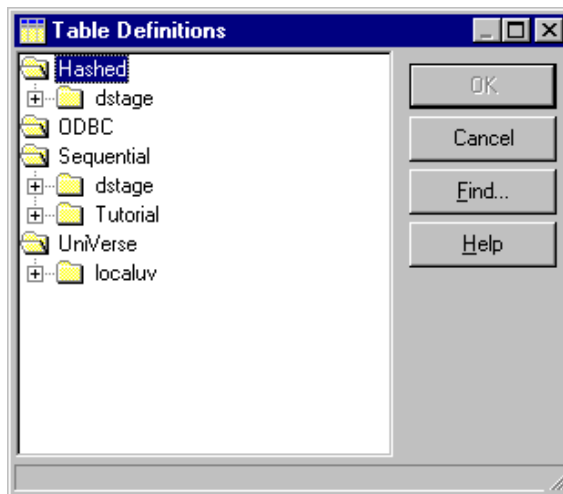
You can continue to add more column definitions by editing the last row in the grid. New column definitions are always added to the bottom of the grid, but you can select and drag the row to a new position in the grid.

12. Click **OK** to save the column definitions and to close the Table Definition dialog box.

### Loading Column Definitions

Instead of entering column definitions, you can load (copy) the column definitions from an existing table definition. To load column definitions:

1. Click **Load...**. The Table Definitions window appears:



This window displays all the table definitions in the project in the form of a table definition tree.

2. Double-click the appropriate branch to display the table definitions available.
3. Select the table definition you want to use.

**Note:** You can use the **Find...** button to enter the name of the table definition you want. The table definition is automatically highlighted in the tree when you click **OK**.

4. Click **OK**. The column definitions from the chosen table definition are copied into your new table definition.
5. Save the table definition by clicking **OK**.

You can edit the table definition to remove unwanted column definitions, assign data elements, or change branch names.

## Viewing or Modifying a Table Definition

You can view or modify any table definition in your project. To view a table definition, select it in the display area and do one of the following:

- Choose **File ► Properties...**
- Choose **Properties...** from the shortcut menu.
- Double-click the table definition in the display area.
- Click the **Properties** icon on the toolbar.

The Table Definition dialog box appears. You can edit any of the column definition properties or delete unwanted definitions.

## Editing Column Definitions

To edit a column definition in the grid, click the cell you want to change. The way you edit the cell depends on the cell contents. If the cell contains a drop-down list, choose an alternative option from the drop-down list. If the cell contains text, you can start typing to change the value, or press **F2** or choose **Edit cell...** from the shortcut menu to put the cell into edit mode.

For more information about adding and deleting rows, or moving between the cells of a grid, see Appendix B, “Editing Grids.”

## Deleting Column Definitions

When you import a table definition, all the column definitions in the data table are imported. Also, when you load column definitions, all the column definitions in the existing table definition are copied.

If you do not want to extract the data in a particular column, or do not want to write data to a particular column in the data mart, you must delete the corresponding column definition.

Unwanted column definitions can be easily removed from the Columns grid. To delete a column definition, click any cell in the row you want to remove and press the **Delete** key or choose **Delete row** from the shortcut menu. Click **OK** to save any changes and to close the Table Definition dialog box.

To delete several column definitions at once, hold down the **Ctrl** key and click in the row selector column for the rows you want to remove. Press the **Delete** key or choose **Delete row** from the shortcut menu to remove the selected rows.

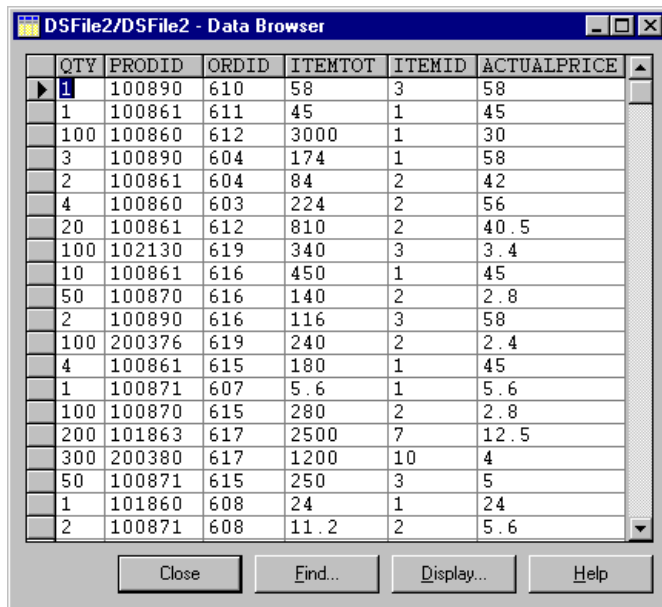
## Using the Data Browser

When importing table definitions from a data source, you can view the actual data in the tables using the Data Browser. The Data Browser can be used when importing table definitions from the following sources:

- ODBC table
- UniVerse table
- Hashed (UniVerse) file
- UniData file
- Some types of plug-in

**Note:** Sequential files can be viewed through standard Windows tools, such as WordPad.

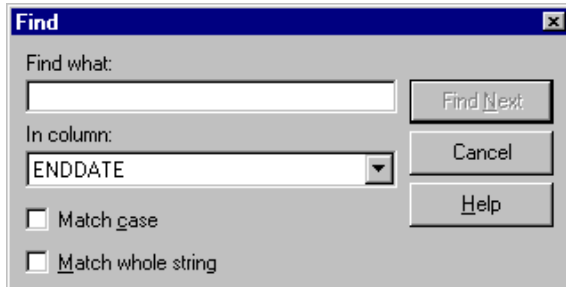
The Data Browser is invoked by clicking the **View Data...** button on the Import Metadata dialog box. The Data Browser window appears:



	QTY	PRODID	ORDID	ITEMTOT	ITEMID	ACTUALPRICE
▶	1	100890	610	58	3	58
	1	100861	611	45	1	45
	100	100860	612	3000	1	30
	3	100890	604	174	1	58
	2	100861	604	84	2	42
	4	100860	603	224	2	56
	20	100861	612	810	2	40.5
	100	102130	619	340	3	3.4
	10	100861	616	450	1	45
	50	100870	616	140	2	2.8
	2	100890	616	116	3	58
	100	200376	619	240	2	2.4
	4	100861	615	180	1	45
	1	100871	607	5.6	1	5.6
	100	100870	615	280	2	2.8
	200	101863	617	2500	7	12.5
	300	200380	617	1200	10	4
	50	100871	615	250	3	5
	1	101860	608	24	1	24
	2	100871	608	11.2	2	5.6

The Data Browser uses the meta data defined in the data source. If there is no data, a Data source is empty message appears instead of the Data Browser.

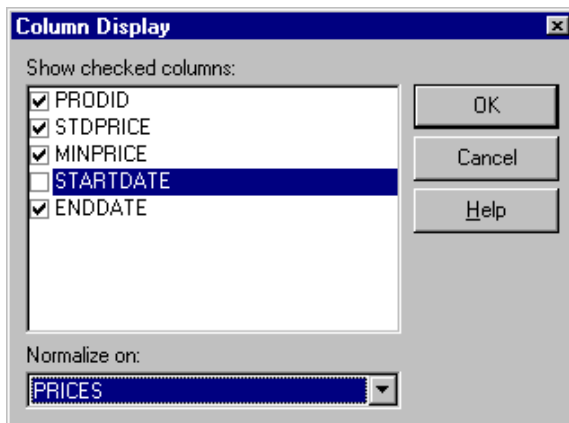
You can view a row containing a specific data item using the **Find...** button. The Find dialog box repositions the view to the row containing the data you are interested in. The search is started from the current row.



The **Display...** button invokes the Column Display dialog box. This allows you to simplify the data displayed by the Data Browser by choosing to hide some of the columns. It also allows you to normalize multivalued data to provide a 1NF view in the Data Browser.

This dialog box lists all the columns in the display, and initially these are all checked. To hide a column, uncheck it.

The Normalize on drop-down list box allows you to select an association or an unassociated multivalued column on which to normalize the data. The default is **Un-Normalized**, and choosing **Un-Normalized** will display the data in NF<sup>2</sup> form with each row shown on a single line. Alternatively you can select **Un-Normalize (formatted)**, which displays multivalued rows split over several lines.



In the example, the Data Browser would display all columns except **STARTDATE**. The view would be normalized on the association **PRICES**.

## Stored Procedure Definitions

If you are accessing data from or writing data to a database via an ODBC connection, you can use a stored procedure to define the data to use. A stored procedure can:

- Have associated parameters, which may be input or output
- Return a value (like a function call)
- Create a result set in the same way as an SQL SELECT statement

**Note:** DataStage supports the use of stored procedures with or without input arguments and the creation of a result set, but does not support output arguments or return values. A stored procedure may have a return value defined, but it is ignored at run time. A stored procedure may not have output parameters.

The definition for a stored procedure (including the associated parameters and meta data) can be stored in the Repository. These stored procedure definitions can be used when you edit an ODBC or UniVerse stage in your job design. For more information about developing a job, see Chapter 5, “Developing a Job.” For more information about the use of stored procedures in UniVerse and ODBC stages, see Chapter 7, “UniVerse Stages” and Chapter 6, “ODBC Stages.”

You can import, create, or edit a stored procedure definition using the DataStage Manager.

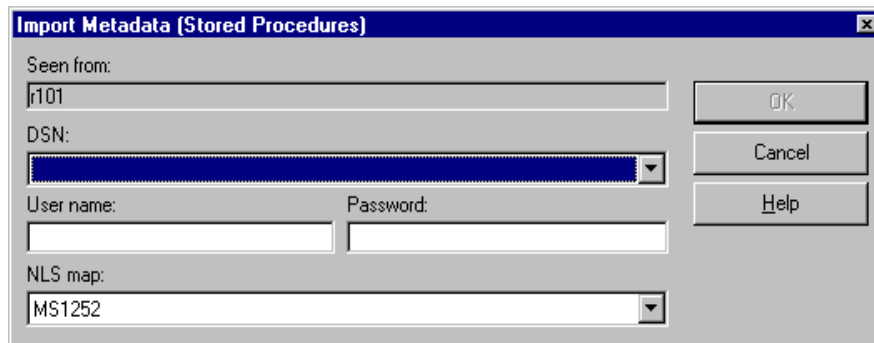
### Importing a Stored Procedure Definition

The easiest way to specify a definition for a stored procedure is to import it directly from the stored procedure on the source or target ODBC database. A new definition for the stored procedure is created and stored in the Repository.

To import a definition for a stored procedure via an ODBC connection:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.

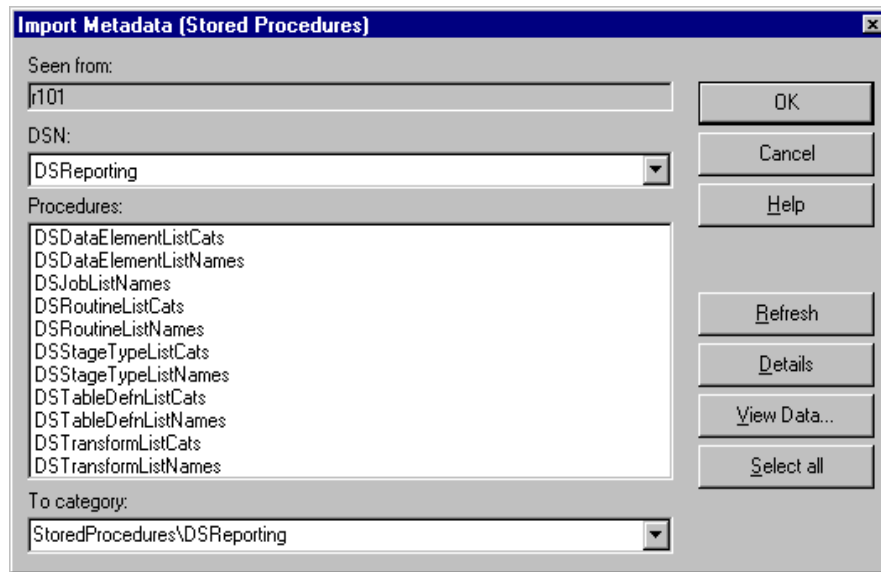
2. Choose **Tools** ► **Import** ► **Stored Procedure Definitions...** . The Import Metadata (Stored Procedures) dialog box appears:



3. Choose a data source name from the DSN drop-down list box. This list box contains the system DSNs defined on the server. For more information about defining ODBC data sources, see *DataStage Administrator's Guide*.
4. Enter your user name in the **User name** field.
5. Enter your password in the **Password** field.  
**Note:** The user name and password are those required to access the system specified by the DSN and might not be the same as those used to access DataStage. Also note that the system specified by the DSN might not be the same as the one named in the **Seen from** field.
6. If NLS is enabled, and you do not want to use the default project map, choose an NLS character set map from the NLS map drop-down list. This map defines the character set that the stored procedure uses.



7. Click **OK**. A connection is made to the ODBC data source and the updated Import Metadata (Stored Procedures) dialog box displays the available stored procedures:



The **To category** field also displays the branch (under the main **Table Definitions** branch) in the Repository where the definition will be stored. This is always the **StoredProcedures ► DSN** branch for stored procedures. *DSN* is the data source name you are importing from.

8. Select a procedure or procedures from the Procedures list box or click **Select all** to choose all available procedures.

You can use **Details** to display further information about each procedure. The displayed information includes the fully qualified procedure name, the names of the associated parameters, and the columns in the result set.

Use **Refresh** to update the Procedures list box.

9. Click **OK**. The definition for the chosen procedure or procedures is imported and a new definition is created under the **Table Definitions ► StoredProcedures** branch in the project tree.

**Note:** If the stored procedure you chose contains output parameters, an error message appears and the import fails. If the stored procedure has a return value, a message appears to warn that this value will be ignored.

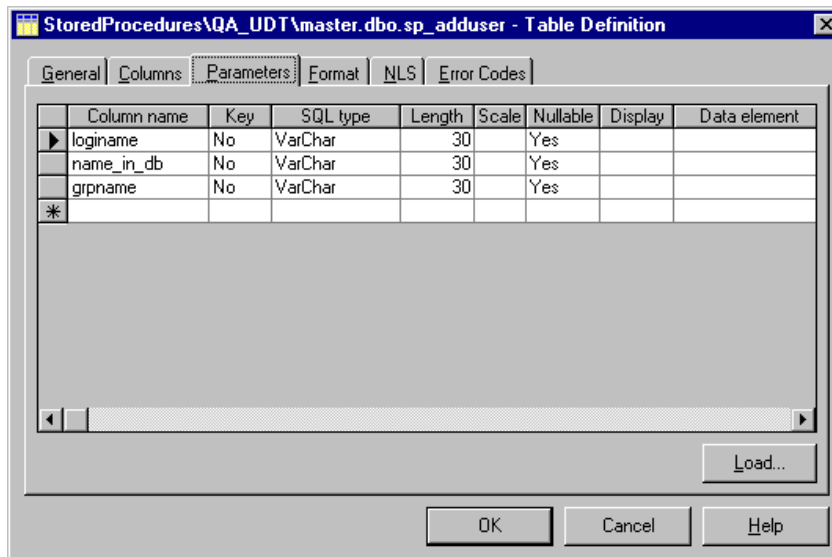
## The Table Definition Dialog Box for Stored Procedures

When you create, edit, or view a stored procedure definition, the Table Definition dialog box appears. This dialog box is described under “The Table Definition Dialog Box” on page 4-23.

The dialog box for a stored procedure has up to five pages:

- **General.** Contains general information about the stored procedure. The **Data source type** field on this page must contain **StoredProcedures** to display the additional **Parameters** page.
- **Columns.** Contains a grid displaying the column definitions for each column in the stored procedure result set. You can add new column definitions, delete unwanted definitions, or edit existing ones. For more information about editing a grid, see Appendix B, “Editing Grids.”
- **Parameters.** Contains a grid displaying the properties of each input parameter.

**Note:** If you cannot see the **Parameters** page, you must enter **StoredProcedures** in the **Data source type** field on the **General** page.



The grid has the following columns:

- **Column name.** The name of the parameter column.

- **Key.** Indicates whether the column is part of the primary key.
- **SQL type.** The SQL data type.
- **Length.** The data precision. This is the length for CHAR data and the maximum length for VARCHAR data.
- **Scale.** The data scale factor.
- **Nullable.** Specifies whether the column can contain null values.
- **Display.** The maximum number of characters required to display the column data.
- **Data element.** The type of data in the column.
- **Description.** A text description of the column.
- **Format.** Contains file format parameters for sequential files. This page is not used for a stored procedure definition.
- **NLS.** Contains the name of the character set map to use with the table definitions.

## Manually Entering a Stored Procedure Definition

If you are unable to import the definition for your stored procedure, you must enter this information manually.

To manually enter a stored procedure definition, first create the definition using the DataStage Manager. You can then enter suitable settings for the general properties, before specifying definitions for the columns in the result set and the input parameters.

**Note:** You do not need to edit the **Format** page for a stored procedure definition.

## Creating a Stored Procedure Definition

To create a stored procedure definition:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Select the **Table Definitions** branch and do one of the following:
  - Choose **File ► New Table Definition...**
  - Choose **New Table Definition...** from the shortcut menu.
  - Click the **New** icon on the toolbar.

The Table Definition dialog box appears. You must enter suitable values in the fields on the **General** page.

3. Enter **StoredProcedures** in the **Data source type** field. This specifies that the new definition will be stored under the **Table Definitions ► StoredProcedures** branch in the Repository. The **Parameters** page appears in the Table Definition dialog box.
4. Enter the name of the data source in the **Data source name** field. This forms the second part of the table definition identifier and is the name of the branch created under the data source type branch.
5. Enter the name of the procedure in the **Procedure name** field. This is the last part of the table definition identifier and is the name of the leaf created under the data source branch.
6. Optionally enter a brief description of the data in the **Short description** field. The text entered here is displayed when you choose **View ► Details** from the DataStage Manager window.
7. Optionally enter a detailed description of the data in the **Long description** field.
8. Specify the column definitions for the result set on the **Columns** page and the input parameters (if any) for the stored procedure on the **Parameters** page.
9. If NLS is enabled and you do not want to use the default project NLS map, select the required map from the **NLS** page.

### **Specifying Column Definitions for the Result Set**

To specify the column definitions for the result set, click the **Columns** tab in the Table Definition dialog box. The **Columns** page appears at the front of the Table Definition dialog box. You can now enter or load column definitions. For more information, see “Entering Column Definitions” on page 4-29 and “Loading Column Definitions” on page 4-31.

**Note:** You do not need a result set if the stored procedure is used for input (writing to a database). However, in this case, you must have input parameters.

### **Specifying Input Parameters**

To specify input parameters for the stored procedure, click the **Parameters** tab in the Table Definition dialog box. The **Parameters** page appears at the front of the Table Definition dialog box.

Parameter definitions are entered directly in the Parameters grid. To add a new parameter definition, you must edit the empty row (indicated by a star) in the Parameters grid:

1. Click the first cell in the row (**Column name**) and enter the name of the parameter you want to add. This is the minimum requirement for adding a new parameter definition. Click the other cells you want to edit, or press **Tab** to move between the cells. For more information about adding and deleting rows, or moving between the cells of a grid, see Appendix B, “Editing Grids.”
2. Specify the SQL data type by choosing an appropriate type from the drop-down list in the **SQL type** cell.
3. Enter an appropriate value for the data precision in the **Length** cell.
4. Enter an appropriate data scale factor in the **Scale** cell.
5. Specify whether the parameter can contain null values by choosing an appropriate option from the drop-down list in the **Nullable** cell.
6. Enter the maximum number of characters required to display the parameter data in the **Display** cell.
7. Choose the type of data the column contains from the drop-down list in the **Data element** cell. This list contains all the built-in data elements supplied with DataStage and any additional data elements you have defined. You do not need to edit this cell to create a column definition. You can assign a data element at any point during the development of your job. For more information, see “Assigning Data Elements” on page 4-46.
8. Enter text to describe the column in the **Description** cell. This cell expands to a drop-down text entry box if you enter more characters than the display width of the column. You can increase the display width of the column if you want to see the full text description.

You can continue to add more parameter definitions by editing the last row in the grid. New parameters are always added to the bottom of the grid, but you can select and drag the row to a new position in the grid.

9. Click **OK** to save and close the Table Definition dialog box.

## Specifying NLS Mapping

If NLS is enabled and you want to use a different character set map than that defined as the project default, click the **NLS** tab in the Table Definition dialog box. The **NLS** page appears at the front of the Table Definition dialog box. Choose the name of the map to use from the list box. By default, the list box shows all the maps that are loaded and ready to use. **Show all maps** lists all the maps that are shipped with DataStage. Click **Allow per-column mapping** if you want to assign different character set maps to individual columns.

**Note:** You cannot use a map unless it is loaded into DataStage. You can load different maps using the DataStage Administrator. For more information, see *DataStage Administrator's Guide*.

## Viewing or Modifying a Stored Procedure Definition

You can view or modify any stored procedure definition in your project. To view a stored procedure definition, select it in the display area and do one of the following:

- Choose **File ► Properties...**
- Choose **Properties...** from the shortcut menu.
- Double-click the stored procedure definition in the display area.
- Click the **Properties** icon on the toolbar.

The Table Definition dialog box appears. You can edit or delete any of the column or parameter definitions.

## Editing Column or Parameter Definitions

You can edit the settings for a column or parameter definition by editing directly in the Columns or Parameters grid. To edit a definition, click the cell you want to change. The way you edit the cell depends on the cell contents. If the cell contains a drop-down list, choose an alternative option from the drop-down list. If the cell contains text, you can start typing to change the value, or press **F2** or choose **Edit cell...** from the shortcut menu to put the cell into edit mode.

For more information about adding and deleting rows, or moving between the cells of a grid, see Appendix B, "Editing Grids."

## Deleting Column or Parameter Definitions

When you import a stored procedure definition, all the column definitions specified for the result set are imported. Also, when you load column definitions, all the column definitions in the existing table definition are copied.

If you do not want to extract or write the data in a particular column or use a particular input parameter, you must delete the corresponding definition.

Unwanted column or parameter definitions can be easily removed from the Columns or Parameters grid. To delete a column or parameter definition, click any cell in the row you want to remove and press the **Delete** key or choose **Delete row** from the shortcut menu. Click **OK** to save any changes and to close the Table Definition dialog box.

To delete several column or parameter definitions at once, hold down the **Ctrl** key and click in the row selector column for the rows you want to remove. Press the **Delete** key or choose **Delete row** from the shortcut menu to remove the selected rows.

## Data Elements

Each column definition may have a data element assigned to it. A *data element* specifies the type of data a column contains, which in turn determines the transforms that can be applied in a Transformer stage.

You can choose to use any of the data elements supplied with DataStage, or you can create and use data elements specific to your application. For a list of the built-in data elements, see “Built-In Data Elements” on page 4-47. You can also import data elements from another data warehousing tool using a MetaBroker. For more information, see Chapter 20, “Using MetaBrokers.”

Application-specific data elements allow you to describe the data in a particular column in more detail. The more information you supply to DataStage about your data, the more DataStage can help to define the processing needed in each Transformer stage.

For example, if you have a column containing a numeric product code, you might assign it the built-in data element **Number**. There is a range of built-in transforms associated with this data element. However, all of these would be unsuitable, as it is unlikely that you would want to perform a calculation on a product code. In this case, you could create a new data element called **PCode**.

Each data element has its own specific set of transforms which relate it to other data elements. When the data elements associated with the columns of a target table are not the same as the data elements of the source data, you must ensure that you have the transforms needed to convert the data as required. For each target column, you should have either a source column with the same data element, or a source column that you can convert to the required data element.

For example, suppose that the target table requires a product code using the data element **PCode**, but the source table holds product data using an older product numbering scheme. In this case, you could create a separate data element for old-format product codes called **Old\_PCode**, and you then create a custom transform to link the two data elements; that is, its source data element is **Old\_PCode**, while its target data element is **PCode**. This transform, which you could call **Convert\_PCode**, would convert an old product code to a new product code. See “Transforms” on page 11-15 for details on how to use and create custom transforms.

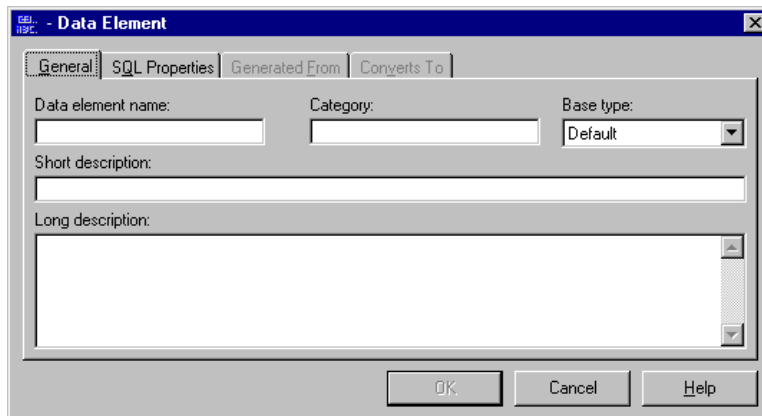
A data element can also be used to “stamp” a column with SQL properties when you manually create a table definition or define a column definition for a link in a job. For details on how to edit the column definition for a link, see “Editing Column Definitions” on page 5-20.

## Creating Data Elements

To create a data element:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Select the **Data Elements** branch in the project tree and do one of the following:
  - Choose **File ► New Data Element...**
  - Choose **New Data Element...** from the shortcut menu.
  - Click the **New** icon on the toolbar.

The Data Element dialog box appears:





This dialog box has four pages:

- **General.** Displayed by default. Contains general information about the data element.
  - **SQL Properties.** Contains fields that describe the properties of the associated SQL data type. This page is used when this data element is used to manually create a new column definition for use with an SQL data source. If you import the column definition from an SQL data source, the SQL properties are already defined.
  - **Generated From.** Lists the transforms that result in this data element.
  - **Converts To.** Lists the transforms that can be used to convert this data element into another data element.
3. Enter the name of the data element in the **Data element name** field. This name is used to create a leaf under the category branch. The name entered here must be unique as no two data elements can have the same name.
  4. Enter a category name in the **Category** field. This name is used to create a branch under the main **Data Elements** branch. If you do not enter a name in this field, the data type is created under the main **Data Elements** branch.

You can create more than one branch level for the category by including \ in the name. For example, if you enter **Custom\Elements**, the following branches are created:

**Data Elements ► Custom ► Elements**

The new data element is stored as a leaf under the **Elements** branch.

5. Choose the most appropriate base data type from the Base type drop-down list box. The base types are the fundamental data types used internally by DataStage for processing.

There are five base types:

- **Date.** The column contains a date, represented in DataStage internal format. There are many built-in transforms available to convert dates to character strings. See Appendix A, “Built-In Transforms and Routines,” for a list of the built-in transforms available.
- **Number.** The column contains a numeric value.
- **String.** The column contains data as a string of characters. DataStage interprets the string as a number if needed.
- **Time.** The column contains data as a time.

- **Default.** The data has an SQL data type already assigned and the most appropriate base type is used.
6. Optionally enter a brief description of the data in the **Short description** field. The text entered here is displayed when you choose **View ► Details** from the DataStage Manager window or print a data elements report.
  7. Optionally enter a detailed description of the data in the **Long description** field. This description is displayed only when you view the properties of a data element.
  8. Click **OK** to save the data element and to close the Data Element dialog box. You must edit your table definition to assign this new data element.

## Assigning Data Elements

If you created a new data element or you want to use one of the data elements supplied with DataStage, you need to assign it. Data elements are assigned by editing the column definitions. If you want to set up the data elements before you develop your job, you can edit the column definitions in the table definition.

To assign a data element:

1. Start the DataStage Manager and enter your project details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Expand the **Table Definitions** branch until you can see the table definition you want to edit in the display area.
3. Select the table definition and do one of the following:
  - Choose **File ► Properties...**
  - Choose **Properties...** from the shortcut menu.
  - Double-click the table definition in the display area.
  - Click the **Properties** icon on the toolbar.

The Table Definition dialog box appears.

4. Click the **Columns** tab. The **Columns** page appears at the front of the Table Definition dialog box.
5. Click the **Data element** cell for the column definition you want to edit.
6. Choose the data element you want to use from the drop-down list. This list contains all the built-in data elements supplied with DataStage and any data elements you created. For a description of the built-in data elements supplied with DataStage, see “Built-In Data Elements” on page 4-47.

7. Click **OK** to save the column definition and to close the Table Definition dialog box.

**Note:** You can assign data elements at any point during the development of your job. For details on how to edit column definitions, see “Editing Column Definitions” on page 5-20.

## Viewing or Editing Data Elements

You can view the properties of any data element in your project. To view the properties of a data element, select it in the display area and do one of the following:

- Choose **File ► Properties...**
- Choose **Properties...** from the shortcut menu.
- Double-click the data element in the display area.
- Click the **Properties** icon on the toolbar.

The Data Element dialog box appears. Click **OK** to close the dialog box.

If you are viewing the properties of a data element that you created, you can edit any of the fields on the **General** or **SQL Properties** page. The changes are saved when you click **OK**.

If you are viewing the properties of a built-in data element, you cannot edit any of the settings on the **General** or **SQL Properties** page.

## Built-In Data Elements

DataStage has a number of built-in data elements. There are six data elements that represent each of the base types used internally by DataStage:

- **Date.** The column contains a date, represented in DataStage internal format. There are many built-in transforms available to convert dates to character strings.
- **Number.** The column contains a numeric value.
- **String.** The column contains data as a string of characters. DataStage interprets the string as a number if needed.
- **Time.** The column contains data as a time.
- **Default.** The data has an SQL data type already assigned and the most appropriate base type is used.

- **Timestamp.** The column contains a string that represents a combined date/time:

*YYYY-MM-DD HH:MM:SS*

In addition, there are some data elements that are used to express dates in alternative ways:

- **DATE.TAG.** The data specifies a date and is stored in the following format:  
1993-02-14 (February 14, 1993)
- **WEEK.TAG.** The data specifies a week and is stored in the following format:  
1993W06 (week 6 of 1993)
- **MONTH.TAG.** The data specifies a month and is stored in the following format:  
1993-02 (February 1993)
- **QUARTER.TAG.** The data specifies a quarter and is stored in the following format:  
1993Q1 (quarter 1, 1993)
- **YEAR.TAG.** The data specifies a year and is stored in the following format:  
1993

Each of these data elements has a base type of **String**. The format of the date complies with various ISO 8601 date formats.

You can view the properties of these data elements. You cannot edit them.

## External ActiveX (OLE) Functions

DataStage provides you with the ability to call external ActiveX (OLE) functions which have been installed on the DataStage server. These functions can then be used when you define custom transforms.

To use this facility, you need an automation server that exposes functions via the IDispatch interface and which has an associated type library. This can be achieved via a number of development tools, including Visual Basic.

The first step in using external functions is importing them into the DataStage Repository. The action of importing an external function creates a DataStage routine containing code which calls the external function. The code uses a

DataStage BASIC function that accepts only certain data types. These data types are defined in the *DSOLETYPES.H* file in the *dsinclude* directory for each project and are listed in the DataStage Developer's Help.

Once imported, you can then call the functions when you define a custom transform.

**Note:** This facility is available only on Windows NT servers.

## Importing External ActiveX (OLE) Functions

To import ActiveX (OLE) functions:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools ► Import ► External Function Definitions...** . The Import Transform Functions Definitions Wizard appears and prompts you to supply the pathname of the file containing the transforms to be imported. This is normally a DLL file which must have already been installed on the server machine.
3. Enter or browse for the pathname, then click **Next**. The wizard queries the specified DLL file to establish what automation classes it contains and presents these in a drop-down list.
4. Select an automation class and click **Next**. The wizard interrogates the automation class to obtain details of the suitable functions it supports. It then displays these.
5. Select the functions that you want to import and specify the Repository category under which the functions will appear (the default is **Routines ► class name**). Click **Next**. The wizard displays the details of the proposed import.
6. If you are happy with the details, click **Import**. DataStage starts to generate the required routines and displays a progress bar. On completion a summary screen appears.
7. Click **Finish** to exit the wizard.



# 5

## Developing a Job

This chapter describes the DataStage Designer, DataStage stages, how to develop a job, and how to specify job properties.

A DataStage job populates one or more tables in the target database. There is no limit to the number of jobs you can create in a DataStage project.

Jobs are designed and developed using the DataStage Designer. A job design contains:

- Stages to represent the processing steps required
- Links between the stages to represent the flow of data

Before you start to develop your job, you must plan and set up your project. This is described in Chapter 4, “Setting Up a Project.”

**Note:** If you want to use the reporting tool described in Chapter 18, “Reporting and Printing,” you should ensure that the names of your DataStage components (jobs, stage, links etc.) do not exceed 64 characters.

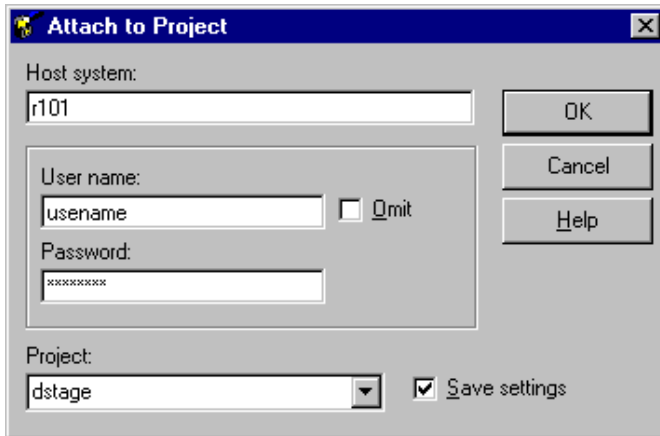
### The DataStage Designer

The DataStage Designer is installed by default in the Ardent DataStage program folder in the Windows Program Manager.

### Starting the DataStage Designer

To start the DataStage Designer, choose **Start** ► **Programs** ► **Ardent DataStage** ► **DataStage Designer**.

The Attach to Project dialog box appears:



You must connect to a project as follows:

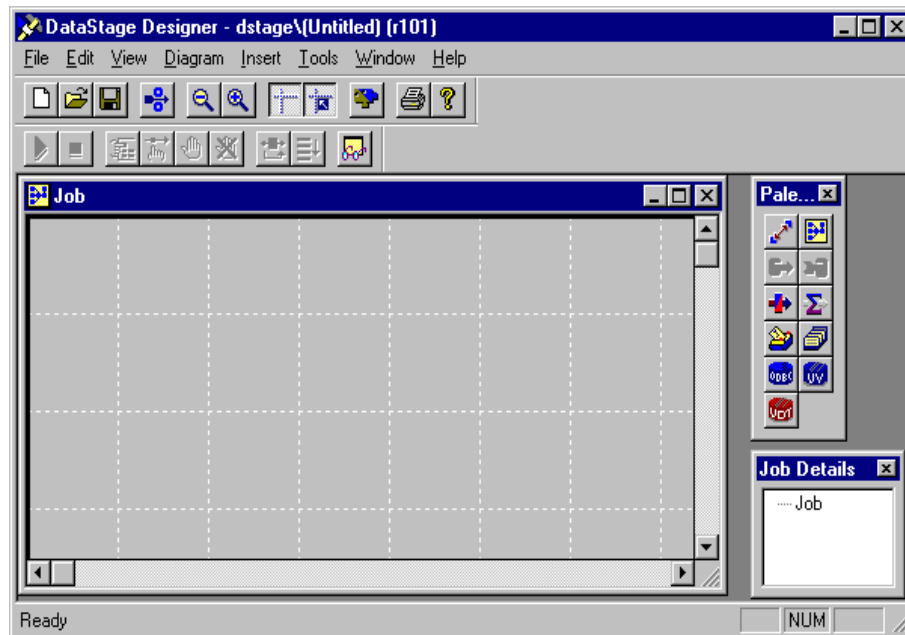
1. Enter the name of your host in the **Host system** field. This is the name of the system where the DataStage Server components are installed.
2. Enter your user name in the **User name** field. This is your user name on the server system.
3. Enter your password in the **Password** field.  
**Note:** If you are connecting to the server via LAN Manager, you can select the **Omit** check box. The **User name** and **Password** fields gray out and you log on to the server using your Windows NT Domain account details.
4. Choose the project to connect to from the Project drop-down list box. This list box displays all the projects installed on your DataStage server.
5. Select the **Save settings** check box to save your logon settings.
6. Click **OK**. The DataStage Designer window appears.

**Note:** You can also start the DataStage Designer from the DataStage Manager or Director by choosing **Tools ► Run Designer**.



## The DataStage Designer Window

The DataStage Designer window appears when you start the Designer:



This window has the following components.

### Menu Bar

There are eight pull-down menus:

- **File.** Creates, opens, and saves DataStage jobs. Also sets up printers, compiles and debugs jobs, and exits the Designer.
- **Edit.** Deletes stages and links in the Diagram window, defines job properties, and displays the stage dialog boxes.
- **View.** Determines what is displayed in the DataStage Designer window. Displays or hides the toolbar, tool palette, Job Details window, debugger toolbar, Watch window, and status bar.

- **Diagram.** Determines what actions are performed in the Diagram window. Displays or hides the grid or print lines, activates or deactivates the **Snap to Grid** option, and zooms in or out of the Diagram window. Also constructs containers.
- **Insert.** Adds a link, built-in stage, or plug-in stage to the Diagram window. The options on this menu perform the same task as the icons on the tool palette.
- **Tools.** Defines the Designer options and starts the DataStage Manager or Director. Also lets you invoke third party applications, or add third party applications to the Designer (see “Customizing the Tools Menu” on page 4-9 for details).
- **Window.** Specifies how the windows are displayed and arranges the icons.
- **Help.** Invokes the Help system. Help is available from all areas of the Designer.

## The Diagram Window

The Diagram window is the canvas on which you design and display your job. By default, this window is empty when you start the Designer allowing you to develop a new job. This window has the following components:

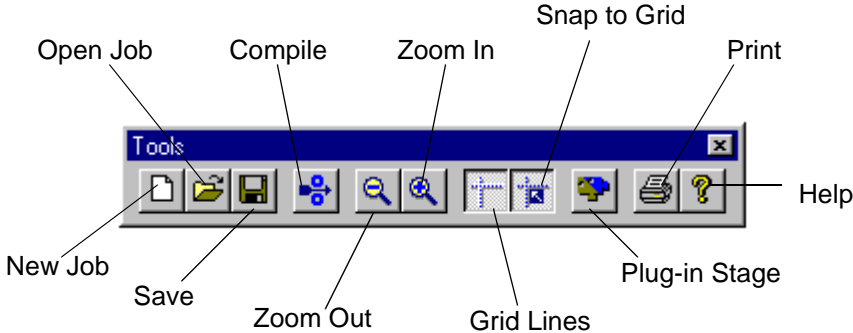
- **Title bar.** Displays “Job” or the name of a container. See “Containers” on page 5-27 for a more detailed description.
- **Grid lines.** Allow you to position stages more precisely in the window. The grid lines are displayed by default. Choose **Diagram ► Show Grid Lines** to disable them.
- **Scroll bars.** Allow you to view the job components that do not fit in the display area.
- **Print lines.** Display the area that is printed when you choose **File ► Print**.

You can use the resize handles or the **Maximize** button to resize the Diagram window. To resize the contents of the window, use the zoom options in the Diagram pull-down menu.

By default, any stages you add to the Diagram window will snap to the grid lines. You can, however, turn this option off by unchecking **Diagram ► Snap to Grid**, clicking the **Snap to Grid** icon in the toolbar, or from the Designer Options dialog box.

**Toolbar**

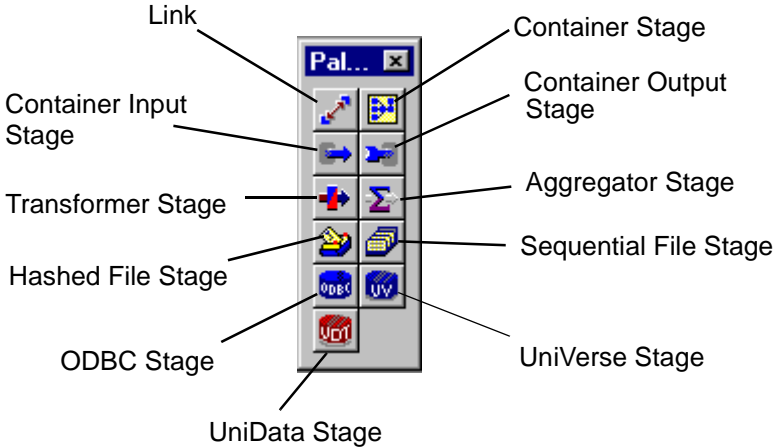
The Designer toolbar contains the following icons:



The toolbar appears under the menu bar by default, but you can drag and drop it anywhere on the screen. If you move the toolbar to the edge of the DataStage Designer window, it attaches to the side of the window. Alternatively, you can hide the toolbar by choosing **View > Toolbar**.

**Tool Palette**

The tool palette contains icons that represent the components you can add to your job design:



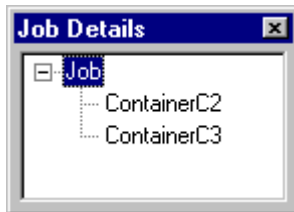
To add a stage to the Diagram window, choose it from the tool palette and click the Diagram window. The stage is added at the insertion point in the Diagram window.

To link two stages, choose the **Link** icon. Click the first stage, then drag the mouse to the second stage. The stages are linked when you release the mouse button.

The tool palette appears in the DataStage Designer window by default, but you can move it anywhere on the screen. You can hide the tool palette by choosing **View ► Palette**.

### The Job Details Window

The Job Details window appears when you choose **View ► Job Details**. If you are using containers in your job design, it lists the containers that make up the job.



Double-click the name of a container in the list to display the component stages in a Diagram window. If you choose a container from the list which is already displayed in a Diagram window, the window appears at the front of the screen.

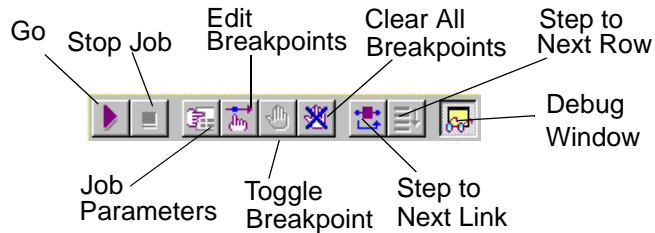
The Job Details window appears by default in the DataStage Designer window, but you can move it anywhere on the screen. You can hide the Job Details window by choosing **View ► Job Details** again.

### Status Bar

The status bar appears at the bottom of the DataStage Designer window. It displays one-line help for the window components and information on the current state of job operations, for example, compilation. You can hide the status bar by choosing **View ► Status Bar**.

## Debugger Toolbar

The debugger toolbar contains icons representing debugger functions. You can hide the debugger toolbar by choosing **View ► Debug Bar**.



## Shortcut Menus

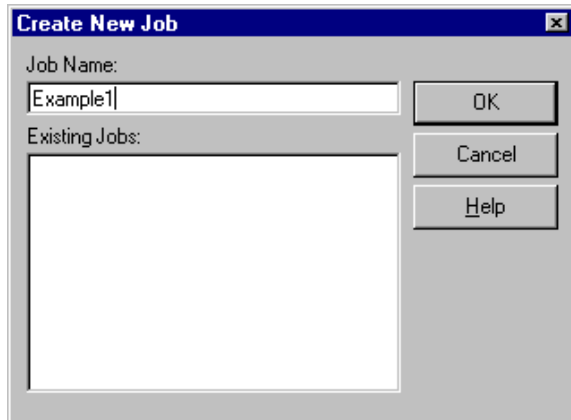
There are four shortcut menus available which you display by clicking the right mouse button. The menu displayed depends on the item you click in the DataStage window.

- **Diagram.** Appears when you click the right mouse button on the window background. This menu contains options to display or hide the grid or print lines, activate or deactivate the **Snap to Grid** option, zoom in or out of the Diagram window, and construct a container.
- **Stage.** Appears when you click the right mouse button on a highlighted stage. This menu contains options to display the stage dialog box and the input or output link properties and to delete the stage. On passive, built-in stages there are also options for invoking the Data Browser on any input or output link meta data.
- **Link.** Appears when you click the right mouse button on a highlighted link. This menu contains options to move or delete the link, change the link type, toggle any breakpoint on the link, or invoke the edit breakpoints dialog box.
- **Job Details.** Appears when you click the right mouse button in the Job Details window. This menu contains options to create, open, save, and compile the job.

## Creating a Job

An empty job is created when you start the Designer. To save the job:

1. Choose **File** ► **Save Job**. The Create New Job dialog box appears:



2. Enter the name of the job in the **Job Name** field.
3. Click **OK**. If the job name is unique, the job is created and saved in the Repository. If the job name is not unique, a message box appears. You must acknowledge this message before you can enter an alternative name.

If you are displaying an existing job design and you want to create a new one, do one of the following:

- Choose **File** ► **New**.
- Click the **New** icon on the toolbar.

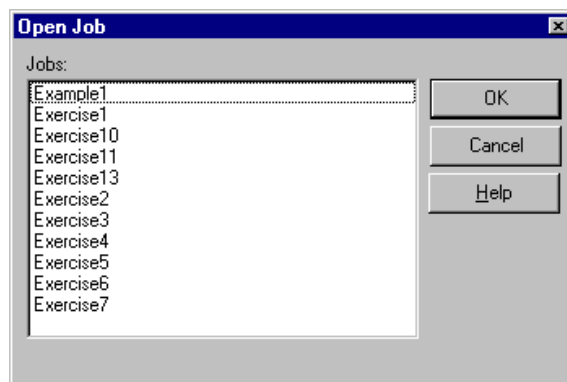
The existing job is closed and the updated DataStage Designer window displays an empty Diagram window. Save the job as described earlier.

## Opening a Job

If you have previously worked on the job you want to open, then you can select it from the list of most recently used jobs in the File menu in the DataStage Designer window. Otherwise, to open a job, do one of the following:

- Choose **File** ► **Open Job...**
- Click the **Open** icon on the toolbar.

The Open Job dialog box appears:



Choose the job you want to open from the Jobs list box and click **OK**.

The DataStage Designer window is updated to display the chosen job in a Diagram window.

## Opening a Job from the DataStage Manager

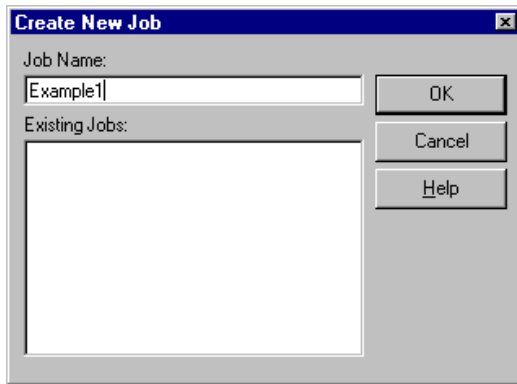
You can also open a job from the DataStage Manager.

1. Select the job to open from the project tree in the DataStage Manager window.
2. Choose **Tools** ► **Run Designer**. The Designer is started and the selected job is opened in the DataStage Designer window.

## Saving a Job

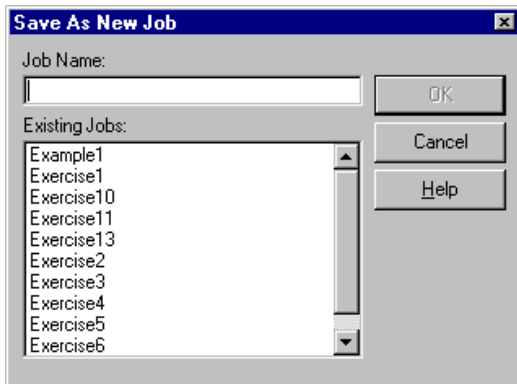
There are two ways of saving jobs in the DataStage Designer:

- **File ► Save Job.** Choose this option to save changes in an open job or to save a new (untitled) job. If you choose this option for a new job, the Create New Job dialog box appears:



Enter a name for the job in the **Job Name** field and click **OK**. The updated title bar in the DataStage Designer window displays the job name.

- **File ► Save Job As...** Choose this option to save an open job as a new job. The Save As New Job dialog box appears:



Enter a name for the job in the **Job Name** field and click **OK**. A copy of the job design is made, a new job is created, and the new job is opened in the Diagram window.



## Exiting the DataStage Designer

To exit the DataStage Designer, choose **File ► Exit** from the DataStage Designer window. If you are currently debugging a job, it is stopped. Your personal settings, such as Diagram window size and position, are preserved and restored when you next use the DataStage Designer.

## Stages

A job consists of stages linked together which describe the flow of data from a data source to a final data warehouse. A stage usually has at least one data input and one data output. However, some stages can accept more than one data input, and output to more than one stage.

DataStage has several built-in stage types that are used to represent data sources, data marts, or conversion stages. These stages are either Passive or Active stages. A Passive stage handles access to databases for the extraction or writing of data. Active stages model the flow of data and provide mechanisms for combining data streams, aggregating data, and converting data from one data type to another.

As well as using the built-in stage types, you can also use plug-in stages for specific operations that the built-in stages do not support.

Stages and links can be grouped together to form a container. A container is represented by a Container stage. The links to and from a container are represented by Container Input and Container Output stages.

Each stage type has a set of predefined and editable properties. These properties are viewed or edited using stage editors. A stage editor exists for each stage type and these are described in detail in individual chapters.

At this point in your job development you need to decide which stage types to use in your job design.

## Built-In Stages

These are the following built-in stage types:

- **Aggregator.** Classifies incoming data into groups, computes totals and other summary functions for each group, and passes them to another stage in the job. This is an active stage.
- **Hashed File.** Extracts data from or loads data into databases that contain hashed files. Also acts as an intermediate stage for quick lookups. This is a passive stage.

- **ODBC.** Extracts data from or loads data into databases that support the industry standard Open Database Connectivity API. This stage is also used as an intermediate stage for aggregating data. This is a passive stage.
- **UniVerse.** Extracts data from or loads data into UniVerse databases. This stage is also used as an intermediate stage for aggregating data. This is a passive stage.
- **UniData.** Extracts data from or loads data into UniData databases. This is a passive stage.
- **Sequential File.** Extracts data from, or loads data into, operating system text files. This is a passive stage.
- **Transformer.** Receives incoming data, transforms it in a variety of ways, and outputs it to another stage in the job. This is an active stage.
- **Container.** Represents a group of stages and links. The group is replaced by a single Container stage in the Diagram window.
- **Container Input and Output.** Represent the interface that links a container stage to the rest of the job design.

Containers, Container stages, and Input and Output stages are described under “Containers” on page 5-27.

## Plug-In Stages

You may find that the built-in stage types do not meet all your requirements for data extraction or transformation. In this case, you need to use a plug-in stage. The function and properties of a plug-in stage are determined by the particular plug-in specified when the stage is inserted. plug-ins are written to perform specific tasks, for example, to bulk load data into a data mart.

There are two plug-ins installed with DataStage: the BCPLoad plug-in and the Orabulk plug-in. These are described in Chapter 14 and Chapter 15 respectively. A range of other plug-ins are supplied with DataStage for you to install if required. Alternatively, you can write your own plug-ins. For more information, see Chapter 13.

## Developing the Job Design

Jobs are designed and developed in the Diagram window. Stages are added and linked together using the tool palette. You can add, move, rename, delete, link, or edit stages in a job design.

## Adding Stages

There is no limit to the number of stages you can add to a job. We recommend you position the stages as follows in the Diagram window:

- Data sources on the left
- Data warehouse on the right
- Transformer stages in the middle of the diagram

Add a stage for each data source and one for the final data mart. Add Transformer stages to convert incoming data and to output the result to another stage in the job. To add a stage:

1. Do one of the following:
  - Choose the stage from the **Insert** menu.
  - Click the stage icon on the tool palette.
2. Click in the Diagram window where you want to position the stage. The stage appears as a square in the Diagram window.

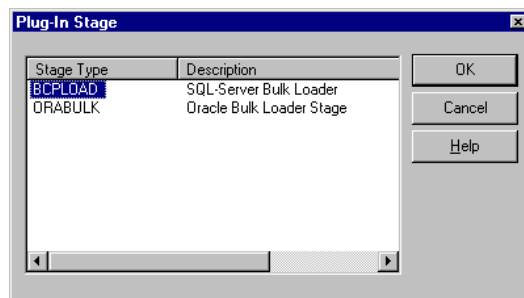
Each stage is given a default name which you can change if required (see “Renaming Stages” on page 5-14).

If you want to add more than one stage of a particular type, press **Shift** before clicking the icon on the tool palette. You can continue to click the Diagram window without having to reselect the icon. Release the **Shift** key when you have added the stages you need; press **Esc** if you change your mind.

## Adding Plug-In Stages

Plug-in stages cannot be added to the Diagram window using the tool palette. To add a plug-in stage:

1. Do one of the following to display the Plug-In Stage dialog box:
  - Choose **Insert ► Plug-in...**
  - Click the **Plug-in Stage** icon on the toolbar.



2. Select the plug-in to use from the list box. This list box contains all the plug-ins defined in your project. The plug-in you choose determines the properties of the plug-in stage.
3. Click **OK**.
4. Click in the Diagram window where you want to position the stage. The stage is added to the Diagram window, which you can edit to define the input and output data.

## Moving Stages

Once positioned, stages can be moved by clicking and dragging them to a new location in the Diagram window. If you have grid lines displayed and the **Snap to Grid** option activated, the stage is attached to the nearest grid position when you release the mouse button. If stages are linked together, the link is maintained when you move a stage.

## Renaming Stages

There are two ways to rename a stage: you can either replace its name entirely or edit the existing name.

To enter a new name:

1. Select the stage in the Diagram window.
2. Type in the new name; as soon as you start typing an edit box appears over the current name showing the characters being typed.
3. Press **Enter** or click outside the stage to accept the edit. The stage name is changed in the Diagram window.

To modify a stage name:

1. Select the stage in the diagram window.
2. Choose **Edit ► Rename**, choose **Rename** from the shortcut menu, or press **Ctrl-R**. An edit box appears over the existing name.
3. Make the required changes in the edit box.
4. Press **Enter** or click outside the stage to accept the edit. The stage name is changed in the Diagram window.

## Deleting Stages

Stages can be deleted from the Diagram window. Choose one or more stages and do one of the following:

- Press the **Delete** key.
- Choose **Edit ► Delete**.
- Choose **Delete** from the shortcut menu.

A message box appears. Click **Yes** to delete the stage or stages and remove them from the Diagram window.

## Links

DataStage supports two types of input link:

- **Stream**. A link representing the flow of data. This is the principal type of link, and is used by both active and passive stages.
- **Reference**. A link representing a table lookup. Reference links are only used by active stages. They are used to provide information that might affect the way data is changed, but do not supply the data to be changed.

The two link types are displayed differently in the Designer Diagram window: stream links are represented by solid lines and reference links by dotted lines. There is only one type of output link, although some stages permit an output link to be used as a reference input to the next stage and some do not.

Built-in stages have maximum and minimum numbers of links as follows:

Stage Type	Stream Inputs		Reference Inputs		Outputs		Reference Outputs?
	Max	Min	Max	Min	Max	Min	
Container	no limit	0	no limit	0	no limit	0	yes
ODBC	no limit	0	0	0	no limit	0	yes
UniVerse	no limit	0	0	0	no limit	0	yes
Hashed File	no limit	0	0	0	no limit	0	yes
UniData	no limit	0	0	0	no limit	0	yes
Sequential File	no limit	0	0	0	no limit	0	no
Transformer	1	1	no limit	0	no limit	1	no
Aggregator	1	1	0	0	no limit	1	no

Plug-in stages supplied with DataStage generally have the following maximums and minimums:

Stage Type	Stream Inputs		Reference Inputs		Outputs		Reference Outputs?
	Max	Min	Max	Min	Max	Min	
Active	1	1	no limit	0	no limit	1	no
Passive	no limit	0	0	0	no limit	0	yes

When designing your own plug-ins, you can specify maximum and minimum inputs and outputs as required.

## Linking Stages

You can link stages in three ways:

- Using the **Link** icon. Choose the **Link** icon from the tool palette. Click the first stage and drag the link to the second stage. The link is made when you release the mouse button.
- Choosing **Insert ► Link**. Click the first stage and drag the link to the second stage. The link is made when you release the mouse button.
- Using the mouse. Select the first stage. Position the mouse cursor on the edge of a stage until the mouse cursor changes to a circle. Click and drag the mouse to the other stage. The link is made when you release the mouse button.

Each link is given a default name which you can change.

## Moving Links

Once positioned, a link (and its associated meta data) can be moved to a new location in the Diagram window. You can choose a new source or destination for the link, but not both.

To move a link:

1. Click the link to move in the Diagram window. The link is highlighted.
2. Click on the box at the end you wish to move and drag the end to its new location.

## Deleting Links

Links can be deleted from the Diagram window. Choose the link and do one of the following:

- Press the **Delete** key.
- Choose **Edit ► Delete**.
- Choose **Delete** from the shortcut menu.

A message box appears. Click **Yes** to delete the link. The link is removed from the Diagram window.

**Note:** Meta data is associated with a link, not a stage. If you delete a link, the associated meta data is deleted too. If you want to retain the meta data you have defined, do not delete the link; move it instead.

## Renaming Links

There are two ways to rename a link; you can either replace its name entirely or edit the existing name.

To enter a new name:

1. Select the link in the Diagram window.
2. Type the new name. As soon as you start typing, an edit box appears over the current name showing the characters being typed.
3. Press **Enter** or click outside the link to accept the edit. The link name is changed in the Diagram window.

To modify a link name:

1. Select the link in the Diagram window.
2. Choose **Edit ► Rename**, choose **Rename** from the shortcut menu, or press **Ctrl-R**. An edit box appears over the existing name.
3. Make the required changes in the edit box.
4. Press **Enter** or click outside the link to accept the edit. The link name is changed in the Diagram window.

## Dealing with Multiple Links

If you have multiple links from one stage to another, you may want to resize the stages in order to make the links clearer by spreading them out. Do this by selecting each stage and dragging on one of the sizing handles in the bounding box.

## Editing Stages

When you have added the stages and links to the Diagram window, you must edit the stages to specify the data you want to use and any aggregations or conversions required.

Data arrives into a stage on an input link and is output from a stage on an output link. The properties of the stage, the data on each input and output link, and any aggregation or transforms required are specified using a stage editor.

To edit a stage, do one of the following:

- Double-click the stage in the Diagram window.
- Select the stage and choose **Properties...** from the shortcut menu.
- Choose **Edit ► Rename**.

A dialog box appears. The content of this dialog box depends on the type of stage you are editing. See the individual stage chapters for a detailed description of the stage dialog box.

The data on a link is specified using column definitions. The column definitions for a link are specified by editing a stage at either end of the link.

Column definitions are entered and edited identically for each stage type.

## Specifying Column Definitions

Each stage dialog box has a page for data inputs and/or data outputs (depending on what links are present on the stage). The data flowing along each input or output link is specified using column definitions. The column definitions are displayed in a grid on the **Columns** page for each link.

The Columns grid has a row for each column definition and always has the following entries:

- **Column name.** The name of the column.
- **Key.** Indicates whether the column is part of the primary key.
- **SQL type.** The SQL data type.
- **Length.** The data precision. This is the length for CHAR data and the maximum length for VARCHAR data.
- **Scale.** The data scale factor.
- **Nullable.** Specifies whether the column can contain null values.
- **Display.** The maximum number of characters required to display the column data.



- **Data element.** The type of data in the column.
- **Description.** A text description of the column.

The grid may contain additional entries, depending on the type of stage you are editing. Specifically, stages that permit aggregating of output data (ODBC and UniVerse stages) also contain the following:

- **Group.** Choose **Yes** to specify that you want to group by this column.
- **Derivation.** Allows you to specify that you want to summarize using this column.

Stages that support multivalued data (UniVerse, UniData, and Hashed File) also contain:

- **Association.** The name of the association (if any) that the column belongs to.
- **Position.** The field number.
- **Type.** The nesting type, which can be S, M, MV, or MS.

Stages that support per-column mapping (Sequential, ODBC, and generic plug-in stages) may contain:

- **NLS Map.** Appears if **Allow per-column mapping** has been selected on the **NLS** page of the stage dialog box. Allows you to choose a separate map for a particular column. This map overrides the default map set for the project, job, stage, or table.

Some entries contain text (which you can edit) and others have a drop-down list containing all the available options for the cell.

You can edit the grid to add new column definitions or change values for existing definitions. For a detailed description of how to edit a grid, see Appendix B, “Editing Grids.” Any changes are saved when you save your job design.

The **Columns** page for each link also contains the following buttons which you can use to edit the column definitions:

- **Save...** . Saves column definitions as a table definition in the Repository.
- **Load...** . Loads (copies) the column definitions from a table definition in the Repository.

## Editing Column Definitions

You can edit the column definitions for any input or output link. Do this by editing the Columns grid.

To edit a column definition, click the cell you want to change. The way you edit the cell depends on the cell contents. If the cell contains a drop-down list, choose an alternative option from the drop-down list. If the cell contains text, you can start typing to change the value, or press **F2** or choose **Edit cell...** from the shortcut menu to put the cell into edit mode.

Some cells are complex, and require more than one piece of information. The **Derivation** column in the Aggregator stage output link is an example of this. In the case of complex cells, editing will invoke a dialog box, allowing you to supply the required information.

## Inserting Column Definitions

If you want to create a new output column or write to a table that does not have a table definition, you can manually enter column definitions by editing the columns grid.

To add a new column at the bottom of the grid, edit the empty row (indicated by a star).

To add a new column between existing rows, position the cursor in the row below the desired position and press the **Insert** key or choose **Insert row...** from the shortcut menu.

In either case, you can select and drag the new row to a new position in the grid.

For more information about adding and deleting rows, or moving between the cells of a grid, see Appendix B, "Editing Grids."

## Deleting Column Definitions

If you have a column definition you do not want, you can delete it. Unwanted column definitions can arise if you load column definitions from a table definition in the Repository (see "Loading Column Definitions" on page 5-22).

To delete a column definition, click anywhere in the row you want to remove and press the **Delete** key or choose **Delete row** from the shortcut menu. The column definition is removed from the grid. Click **OK** to close the Stage dialog box. Changes are saved when you save your job design.

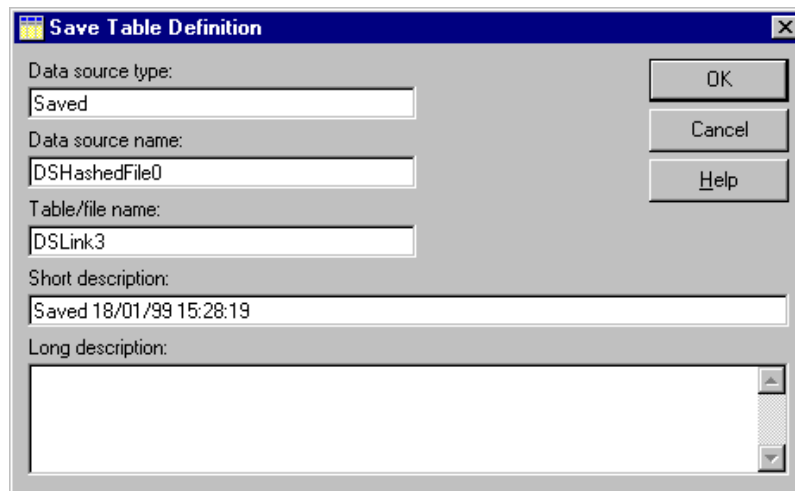
If you want to delete more than one column definition at once, press **Ctrl** and click in the row selector column for the rows you want to remove. Press the **Delete** key or choose **Delete row** from the shortcut menu to remove the selected column definitions.

### Saving Column Definitions

If you edit column definitions or insert new definitions, you can save them in a table definition in the Repository. You can then load the definitions into other stages in your job design.

To save the column definitions:

1. Click **Save...**. The Save Table Definition dialog box appears:



2. Enter a category name in the **Data source type** field. The name entered here determines how the definition will be stored under the main **Table Definitions** branch. By default, this field contains **Saved**.
3. Enter a name in the **Data source name** field. This forms the second part of the table definition identifier and is the name of the branch created under the data source type branch. By default, this field contains the name of the stage you are editing.
4. Enter a name in the **Table/file name** field. This is the last part of the table definition identifier and is the name of the leaf created under the data source name branch. By default, this field contains the name of the link you are editing.

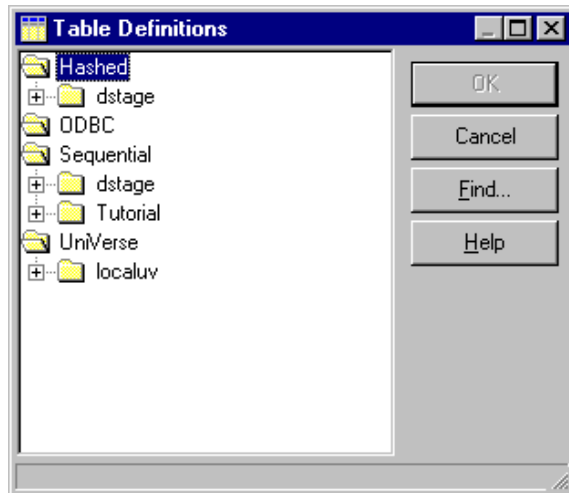
5. Optionally enter a brief description of the table definition in the **Short description** field. By default, this field contains the date and time you clicked **Save...** . The format of the date and time depend on your Windows setup.
6. Optionally enter a more detailed description of the table definition in the **Long description** field.
7. Click **OK**. The column definitions are saved under the specified branches in the Repository.

### Loading Column Definitions

You can load column definitions from a table definition in the Repository. For a description of how to create or import table definitions, see “Table Definitions” on page 4-11.

To load column definitions:

1. Click **Load...** . The Table Definitions window appears. This window displays all the table definitions in your project in the form of a table definition tree. The table definition categories are listed alphabetically in the tree.



2. Double-click the appropriate category branch.
3. Continue to expand the branches until you see the table definition leaves.
4. Select the table definition you want.

**Note:** You can use **Find...** to enter the name of the table definition you want. The table definition is selected in the tree when you click **OK**.

5. Click **OK**. All the column definitions from the chosen table definition are copied into the Columns grid. You can now edit or delete the column definitions.
6. Click **OK** to close the stage dialog box. Changes are saved when you save your job design.

### Browsing Server Directories

When you edit a UniVerse, Sequential File, or Hashed File stage, you may need to specify a directory path on the DataStage server where the required files are found. You can specify a directory path in one of three ways:

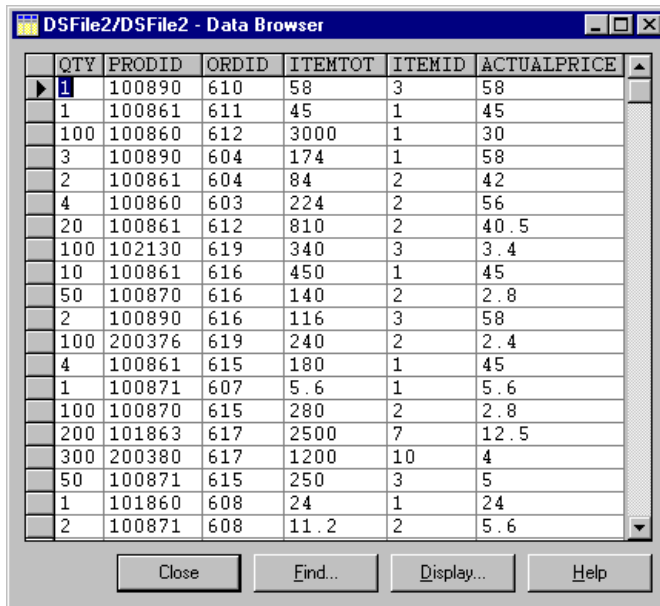
- Enter a job parameter in the respective text entry box in the stage dialog box. For more information about defining and using job parameters, see “Specifying Job Parameters” on page 5-33.
- Enter the directory path directly in the respective text entry box in the Stage dialog box.
- Use **Browse...**

If you use **Browse...**, the Browse directories dialog box appears. This dialog box has the following components:

- **Directory on** field. Displays the directory path. This field is automatically updated with the drive and directory you choose. You can also enter a directory path directly in this field.
- **Directory** list. Displays the directories on the chosen drive. Double-click the directory you want. Double-click **..** to move up a level in the directory structure.
- **Drive on** list (only shown when connected to a Windows NT server). Displays the mounted drives on the DataStage server. Choose the drive you want from the drop-down list. The Directory list box is automatically updated when you choose a drive.
- **OK** button. Accepts the directory path in the **Directory on** field and closes the Browse directories dialog box.
- **Cancel** button. Closes the dialog box without specifying a directory path.
- **Help** button. Invokes the Help system.

## Using the Data Browser

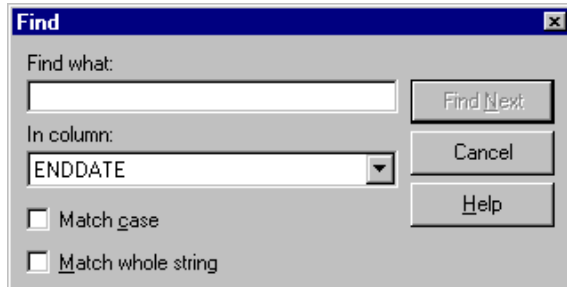
You can browse the data associated with the input or output links of any built-in passive stage. The Data Browser is invoked by clicking the **View Data...** button from a Stage **Inputs** or **Outputs** page, or by choosing the **View link Data...** option from the shortcut menu. The Data Browser window appears:



	QTY	PRODID	ORDID	ITEMTOT	ITEMID	ACTUALPRICE
▶	1	100890	610	58	3	58
	1	100861	611	45	1	45
	100	100860	612	3000	1	30
	3	100890	604	174	1	58
	2	100861	604	84	2	42
	4	100860	603	224	2	56
	20	100861	612	810	2	40.5
	100	102130	619	340	3	3.4
	10	100861	616	450	1	45
	50	100870	616	140	2	2.8
	2	100890	616	116	3	58
	100	200376	619	240	2	2.4
	4	100861	615	180	1	45
	1	100871	607	5.6	1	5.6
	100	100870	615	280	2	2.8
	200	101863	617	2500	7	12.5
	300	200380	617	1200	10	4
	50	100871	615	250	3	5
	1	101860	608	24	1	24
	2	100871	608	11.2	2	5.6

The Data Browser uses the meta data defined for that link. If there is insufficient data associated with a link to allow browsing, the **View Data...** button and shortcut menu option used to invoke the Data Browser are disabled. If the Data Browser requires you to input some parameters before it can determine what data to display, the Job Run Options dialog appears and collects the parameters (see “The Job Run Options Dialog Box” on page 5-26).

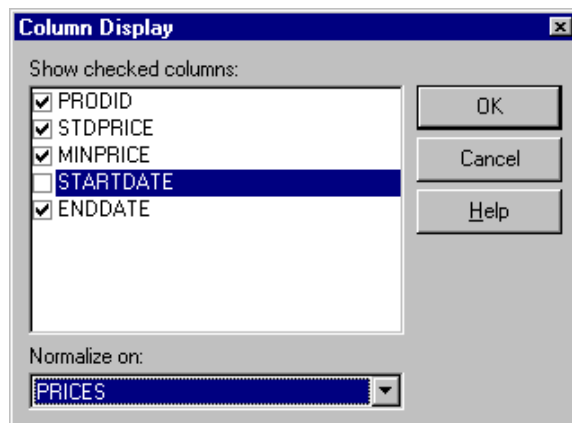
You can view a row containing a specific data item using the **Find...** button. The Find dialog box will reposition the view to the row containing the data you are interested in. The search is started from the current row.



The **Display...** button invokes the Column Display dialog box. This allows you to simplify the data displayed by the Data Browser by choosing to hide some of the columns. It also allows you to normalize multivalued data to provide a 1NF view in the Data Browser.

This dialog box lists all the columns in the display, all of which are initially selected. To hide a column, uncheck it.

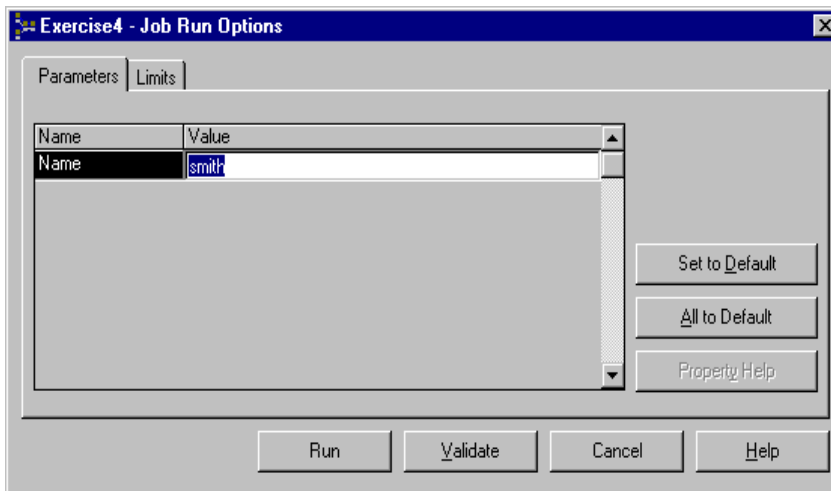
The Normalize on drop-down list box allows you to select an association or an unassociated multivalued column on which to normalize the data. The default is **Un-normalized**, and choosing **Un-normalized** will display the data in NF<sup>2</sup> form with each row shown on a single line. Alternatively you can select **Un-Normalized (formatted)**, which displays multivalued rows split over several lines.



In the example, the Data Browser would display all columns except **STARTDATE**. The view would be normalized on the association **PRICES**.

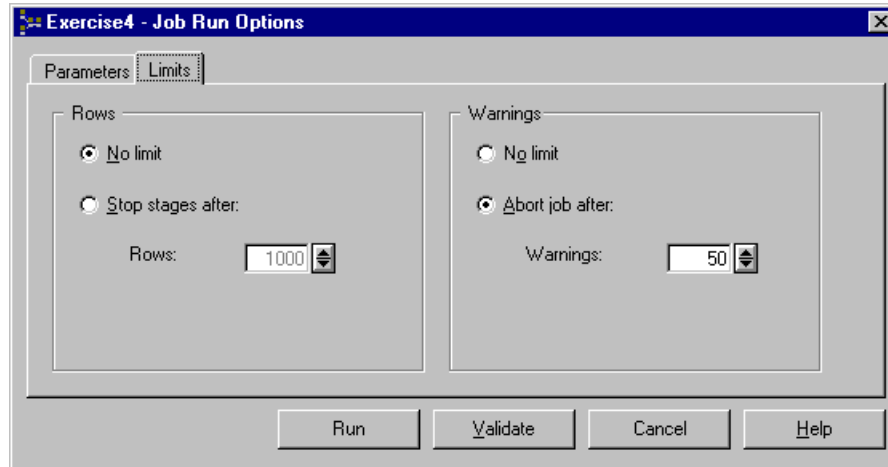
## The Job Run Options Dialog Box

When the DataStage Designer needs you to specify information about the running of a job, it displays the Job Run Options dialog box. This has two pages: one to collect any parameters the job requires and one to let you specify any run-time limits. This dialog box may appear when you are using the Data Browser, specifying a job control routine, or using the debugger.



The **Parameters** page lists any parameters that have been defined for the job. If default values have been specified, these are displayed too. You can enter a value in the **Value** column, edit the default, or accept the default as it is. Click **Set to Default** to set a parameter to its default value, or click **All to Default** to set all parameters to their default values. Click **Property Help** to display any help text that has been defined for the selected parameter (this button is disabled if no help has been defined). Click **OK** when you are satisfied with the values for the parameters.





The **Limits** page allows you to specify whether stages in the job should be limited in how many rows they process and whether run-time error warnings should be ignored.

To specify a row's limits:

1. Click the **Stop stages after** option button.
2. Select the number of rows from the drop-down list box.

To specify that the job should abort after a certain number of warnings:

1. Click the **Abort job after** option button.
2. Select the number of warnings from the drop-down list box.

## Containers

A container is a group of stages and links. A job always consists of one container (the top-level container), which is displayed in the Diagram window when you open a job. If your job consists of only the top-level container, this is the only container listed in the Job Details window.

If the DataStage job has lots of stages and links, it may be easier to create additional containers to describe a particular sequence of steps. The container is displayed in the Diagram window as a Container stage and is listed in the Job Details window.

A job can be made up of any number of containers. A container can contain any number of stages and links. Containers are linked to other stages or containers in the project by input and output stages.

## Creating a Container

If your job design is becoming complex, you can modularize the design by grouping stages and links into a container. To save a group of stages and links in a container:

1. Choose the stages and links by doing one of the following:
  - Click and drag the mouse over all the stages you want in the container.
  - Select a stage. Press **Shift** and click the other stages and links you want to add to the container.

All the chosen stages and links are highlighted in red.

2. Choose **Diagram > Construct Container**. The group is replaced by a Container stage in the Diagram window. A new Designer window is created containing the contents of the new container stage.

## Adding a Container Stage

If you want to describe a series of processing steps in a contained group, you can add a Container stage to your job design. The Container stage is edited to include the stages and links required.

To add a Container stage, do one of the following:

- Choose **Insert > Container**.
- Click the **Container** icon on the tool palette.

A Container stage is added to the Diagram window and the container name is added to the Job Details window.

## Viewing or Modifying a Container

To view or modify the stages or links in a container, do one of the following:

- Double-click the container stage in the Diagram window.
- Double-click the container name in the Job Details window.
- Select the container and choose **Edit > Properties...**
- Select the container and choose **Properties...** from the shortcut menu.

Another Diagram window appears with the contents of the container. You can tile the Diagram windows so that the container is displayed beneath the main Diagram window.

You can edit the stages and links in a container in the same way you do for the top-level container.

Click the **Close** button to close the Diagram window for the container.

## Using Input and Output Stages

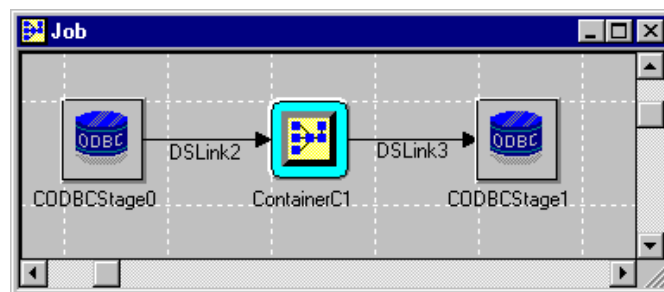
When you use containers in a job design, a link is displayed going in to or out of the container. However, it is not apparent which stage in the container forms the other end of the link.

The link between a stage and a container is represented in the container Diagram window by a Container Input or Output stage.

The way in which the Container Input and Output stages are used depends on whether you create or add a container:

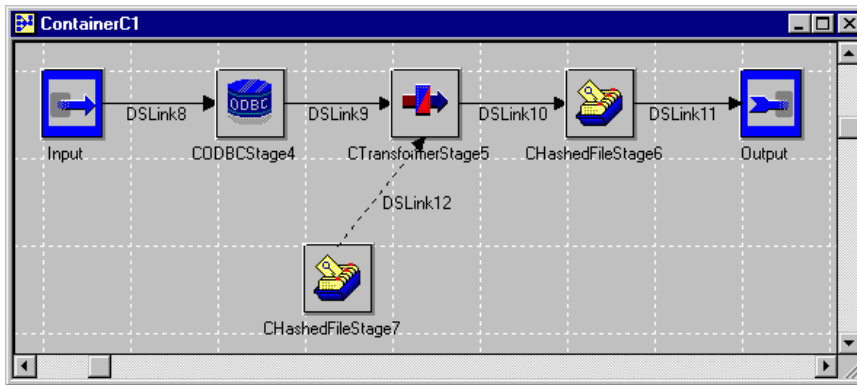
- If you create a container, the input and output stages are automatically added. The link between the input or output stage and the stage in the container has the same name as the link in the main Diagram window.
- If you add a container stage, you must add stages to the container Diagram window between the input and output stages. Link the stages together and edit the link names to match the ones in the main Diagram window.

For example, the following Diagram window shows two ODBC stages linked to a container:



The first ODBC stage links to a stage in the container, which is represented by a Container Input stage. A different stage in the container links to the second ODBC stage, which is represented by a Container Output stage.

The container Diagram window includes the input and output stages required to link to the two ODBC stages. Note that the link names match those used for the links between the ODBC stages and the container in the main Diagram window.

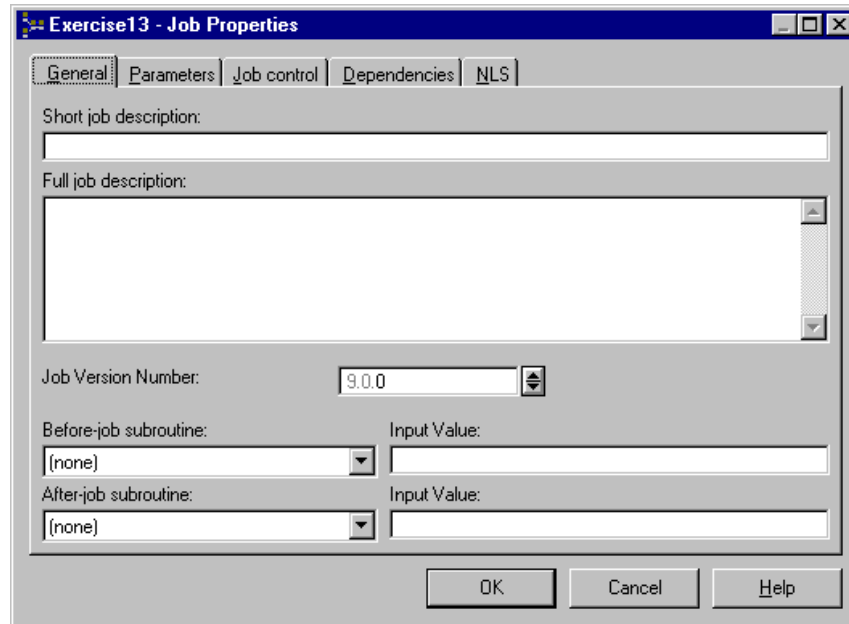


## Job Properties

Each job in a project has properties including optional descriptions and job parameters. You can view and edit the job properties from the DataStage Designer or the DataStage Manager:

- From the Designer. Open the job in the DataStage Designer window and choose **Edit ► Job Properties...**
- From the Manager. Double-click a job in the project tree in the DataStage Manager window or, alternatively, select the job and choose **File ► Properties...**

The Job Properties dialog box appears:



This dialog box has up to five pages: **General**, **Parameters**, **Job control**, **NLS**, and **Dependencies**. The **General** page contains the following fields:

- **Short job description.** An optional brief description of the job.
- **Full job description.** An optional detailed description of the job.
- **Job Version Number.** The version number of the job. A job version number has several components:
  - The version number: *N.n.n*. This number checks the compatibility of the job with the version of DataStage installed. This number is automatically set when DataStage is installed and cannot be edited.
  - The release number: *n.N.n*. This number is automatically incremented every time you release a job. For more information about releasing jobs, see “Releasing a Job” on page 17-7.
  - The bug fix number: *n.n.N*. This number reflects minor changes to the job design or properties. To change this number, select it and enter a new value directly or use the arrow buttons to increase the number.

- **Before-job subroutine** and **Input Value**. Optionally contain the name (and input parameter value) of a subroutine that is executed before the job runs. For example, you can specify a routine that prepares the data before processing starts.

Choose a routine from the drop-down list box. This list box contains all the built routines defined as a **Before/After Subroutine** under the **Routines** branch in the Repository. Enter an appropriate value for the routine's input argument in the **Input Value** field.

If you use a routine that is defined in the Repository, but which was edited and not compiled, a warning message reminds you to compile the routine when you close the Job Properties dialog box.

If you installed or imported a job, the **Before-job subroutine** field may reference a routine which does not exist on your system. In this case, a warning message appears when you close the Job Properties dialog box. You must install or import the "missing" routine or choose an alternative one to use.

- **After-job subroutine** and **Input Value**. Optionally contain the name (and input parameter value) of a subroutine that is executed after the job has finished. For example, you can specify a routine that sends an electronic message when the job finishes.

Choose a routine from the drop-down list box. This list box contains all the built routines defined as a **Before/After Subroutine** under the **Routines** branch in the Repository. Enter an appropriate value for the routine's input argument in the **Input Value** field.

If you use a routine that is defined in the Repository, but which was edited but not compiled, a warning message reminds you to compile the routine when you close the Job Properties dialog box.

If you installed or imported a job, the **After-job subroutine** field may reference a routine that does not exist on your system. In this case, a warning message appears when you close the Job Properties dialog box. You must install or import the "missing" routine or choose an alternative one to use.

For information on the **Parameters**, **Job control**, **NLS**, and **Dependencies** pages, see "Specifying Job Parameters" on page 5-33, "Job Control Routines" on page 5-38, "Specifying Maps and Locales" on page 5-41, and "Specifying Job Dependencies" on page 5-43.

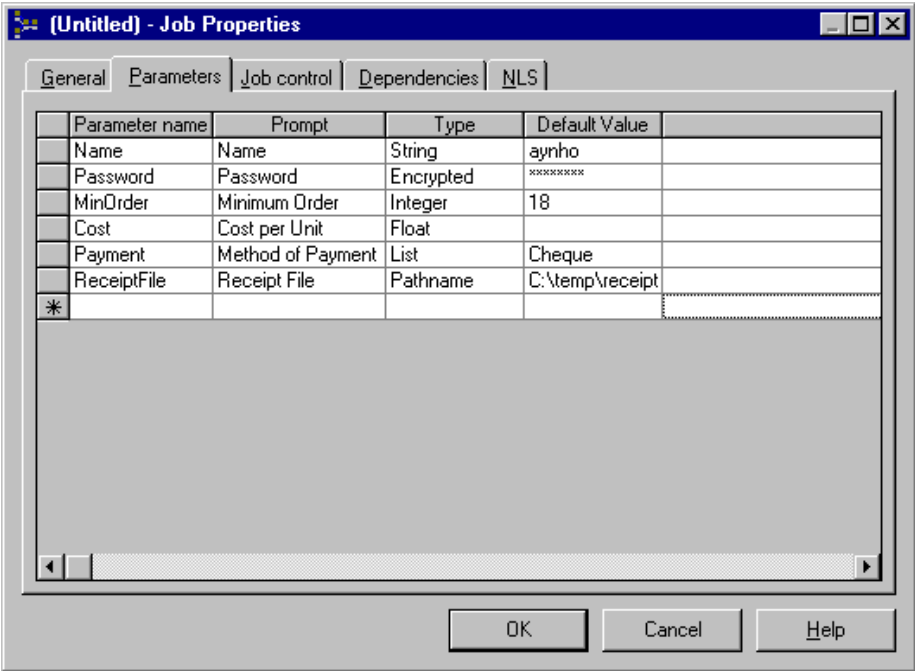
Click **OK** to record your changes in the job design. Changes are not saved to the Repository until you save the job design.

# Specifying Job Parameters

Job parameters allow you to design flexible, reusable jobs. If you want to process data based on the results for a particular week, location, or product you can include these settings as part of your job design. However, when you want to use the job again for a different week or product, you must edit the design and recompile the job.

Instead of entering inherently variable factors as part of the job design you can set up parameters which represent processing variables. Operators are prompted for values when they run or schedule the job.

Job parameters are defined, edited, and deleted in the **Parameters** page of the Job Properties dialog box. For information on how to display this dialog box, see “Job Properties” on page 5-30.



The Job Parameters grid has the following columns:

- **Parameter.** The name of the parameter.
- **Prompt.** Text used as the field name in the run-time dialog box.
- **Type.** The type of the parameter (to enable validation)
- **Default Value.** The default setting for the parameter.
- **Help Text.** The text that appears if a user clicks **Property Help** in the Job Run Options dialog box when running the job.

## Parameter Types

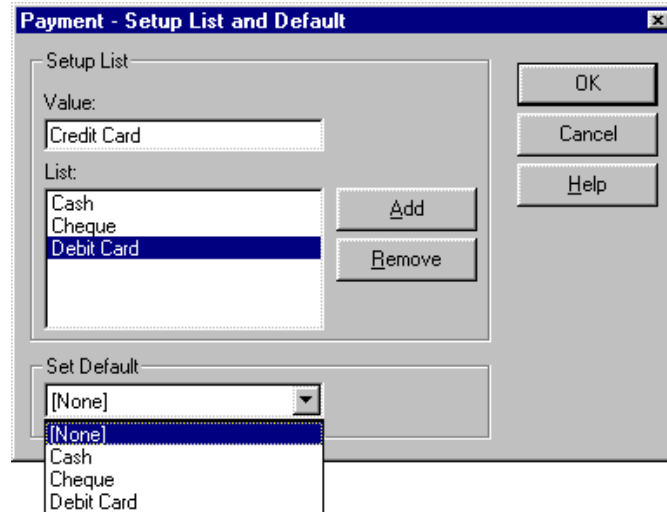
Specify the type of the parameter by choosing one of the following from the drop-down list:

- **String.** The default type.
- **Encrypted.** Used to specify a password. The default value is set by double-clicking the **Default Value** cell to invoke the Setup Password dialog box. Type the password in the **Encrypted String** field and retype it in the **Confirm Encrypted String** field. It is displayed as asterisks.



- **Integer.** Long int (-2147483648 to +2147483647).
- **Float.** Double (1.79769313486232E308 to -4.94065645841247E-324 and 4.94065645841247E-324 to -1.79769313486232E308).
- **Pathname.** Enter a default pathname or filename by typing it into **Default Value** or double-click the **Default Value** cell to invoke the Browse dialog box.
- **List.** A list of valid string variables. To set up a list, double-click the **Default Value** cell to invoke the Set Defaults dialog box. Build up a list by typing in each item into the **Value** field then clicking **Add**. The item then appears in the List box. To remove an item, select it in the List box and click **Remove**. Select one of the items from the Set Default drop-down list box to be the default.





- **Date.** Date in the ISO format *yyy-mm-dd*.
- **Time.** Time in the format *hh:mm:ss*.

DataStage uses the parameter type to validate any values that are subsequently supplied for that parameter, be it in the Director or the Designer.

## Parameter Defaults

You can supply default values for parameters, which are used unless another value is specified when the job is run. For most parameter types, you simply type an appropriate default value into the **Default Value** cell. When specifying a password or a list variable, double-click the **Default Value** cell to invoke further dialog boxes which allow you to supply defaults.

## Defining Job Parameters

Job parameters are defined by editing the empty row (indicated by a star) in the Job Parameters grid. For more information about adding and deleting rows, or moving between the cells of a grid, see Appendix B, “Editing Grids.”

When you have added all the job parameters you need, click **OK** to close the Job Properties dialog box.

## Editing Job Parameters

You can edit any existing job parameter definition. To edit a job parameter, click the cell you want to change. The way you edit the cell depends on the cell contents. If the cell contains a drop-down list, choose an alternative option from the drop-down list.

If the cell contains text, you can start typing to change the value or press **F2** to put the cell into edit mode. When you finish making any changes, click **OK** to close the Job Properties dialog box.

## Deleting Job Parameters

You can delete any of the job parameters defined in the Job Properties dialog box.

**CAUTION:** Before you remove a job parameter definition you must make sure that you remove the references to this parameter in your job design. If you do not do this, your job may fail to run.

To delete a job parameter definition:

1. Click anywhere in the row for the job parameter you want to remove and press the **Delete** key or choose **Delete row** from the shortcut menu. A message box appears.
2. Click **Yes** to delete the job parameter.
3. Click **OK** to close the Job Properties dialog box.

To delete more than one job parameter at once, press **Ctrl** and click in the row selector column for the rows you want to remove, then press the **Delete** key or choose **Delete row** from the shortcut menu. A message box appears. Click **Yes** to delete the parameters.

## Using Job Parameters

To use the defined job parameters, you must specify them when you edit a stage. When you edit a field that you wish to use a parameter for, enter **#Param#**, where *Param* is the name of the job parameter. The string **#Param#** is replaced by the value for the job parameter when the job is run. (For more details about editing stages, see “Editing Stages” on page 5-18.)

A job parameter can be used in any stage or link property, for example:

- In Transformer stages. You can use job parameters in the following fields when you edit a Transformer stage:

- **Key Expression** cell of a reference input link
- **Constraint** on an output link.
- **Derivation** cell of an output link.

You can use the Expression Editor to insert a job parameter in these fields. For information about the Expression Editor, see “The DataStage Expression Editor” on page 11-12.

- In Sequential File stages. You can use job parameters in the following fields in the Sequential File Stage dialog box:
  - **Directory where files are held** field on the **Stage** page
  - **File name** field on the **Inputs** or **Outputs** page
- In ODBC or UniVerse stages. You can use job parameters in the following fields in the Stage dialog box:
  - **Data source name** field on the **General** page on the **Stage** page
  - **User name** and **Password** fields on the **General** page on the **Stage** page
  - **Account name** or **Use directory path** fields on the **Details** page on the **Stage** page (UniVerse stage only)
  - **Table name** field on the **General** page on the **Inputs** or **Outputs** page
  - **WHERE clause** field on the **Selection** page on the **Outputs** page
  - **Value** cell on the **Parameters** page, which appears in the **Outputs** page when you use a stored procedure (ODBC stage only)
  - **Expression** field on the **Derivation** dialog box, invoked from the **Derivation** column in the **Outputs** page of a UniVerse or ODBC Stage dialog box
- In Hashed File stages. You can use job parameters in the following fields in the Hashed File Stage dialog box:
  - **Use account name** or **Use directory path** fields on the **Stage** page
  - **File name** field on the **General** page on the **Inputs** or **Outputs** page

- In UniData stages. You can use job parameters in the following fields in the UniData Stage dialog box:
  - **Server**, **Database**, **User name**, and **Password** fields on the **Stage** page
  - **File name** field on the **General** page on the **Inputs** or **Outputs** page
- Before and after subroutines. You can use job parameters to specify argument values for before and after subroutines.

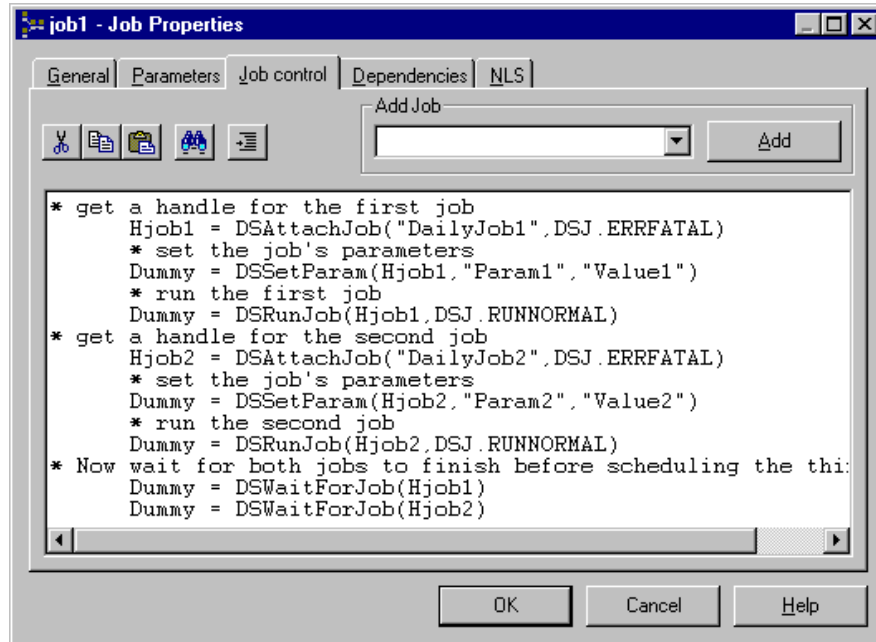
**Note:** You can also use job parameters in the **Property name** field on the **Properties** page in the Stage Type dialog box when you create a plug-in. For more information, see “Manually Registering a Plug-In Definition” on page 13-2.

## Job Control Routines

A job control routine provides the means of controlling other jobs from within the current job. A set of one or more jobs can be validated, run, reset, stopped, and scheduled in much the same way as the current job can be. You can, if required, set up a job whose only function is to control a set of other jobs.

To control other jobs in this way, you must define a job control routine in the **Job control** page of the Job Properties dialog box. The routine uses a set of BASIC functions provided for the purpose. For more information about these routines, see DataStage Developer’s Help. The **Job control** page provides a basic editor to let you construct a job control routine using the functions.

The toolbar contains buttons for cutting, copying, pasting, and formatting code, and for activating **Find** (and **Replace**). The main part of this page consists of a multiline text box with scroll bars. The **Add Job** field provides a drop-down list box of all the jobs in the current project. When you select a compiled job from the list and click **Add**, the Job Run Options dialog box appears, allowing you to specify any parameters or run-time limits to apply when the selected job is run (see “The Job Run Options Dialog Box” on page 5-26). The job will also be added to the list of dependencies (see “Specifying Job Dependencies” on page 5-43). When you click **OK** in the Job Run Options dialog box, you return to the **Job control** page, where you will find that DataStage has added job control code for the selected job. The code sets any required job parameters and/or limits, runs the job, waits for it to finish, then tests for success.



Alternatively, you can type your routine directly into the text box on the **Job control** page, specifying jobs, parameters, and any run-time limits directly in the code.

The following is an example of a job control routine. It schedules two jobs, waits for them to finish running, tests their status, and then schedules another one. After the third job has finished, the routine gets its finishing status.

```
* get a handle for the first job
Hjob1 = DSAttachJob("DailyJob1", DSJ.ERRFATAL)
* set the job's parameters
Dummy = DSSetParam(Hjob1, "Param1", "Value1")
* run the first job
Dummy = DSRunJob(Hjob1, DSJ.RUNNORMAL)

* get a handle for the second job
Hjob2 = DSAttachJob("DailyJob2", DSJ.ERRFATAL)
* set the job's parameters
Dummy = DSSetParam(Hjob2, "Param2", "Value2")
* run the second job
Dummy = DSRunJob(Hjob2, DSJ.RUNNORMAL)

* Now wait for both jobs to finish before scheduling the third job
Dummy = DSWaitForJob(Hjob1)
Dummy = DSWaitForJob(Hjob2)
```

```

* Test the status of the first job (failure causes routine to exit)
J1stat = DSGetJobInfo(Hjob1, DSJ.JOBSTATUS)
If J1stat = DSJS.RUNFAILED
    Then Call DSLogFatal("Job DailyJob1 failed","JobControl")
End

* Test the status of the second job (failure causes routine to exit)
J2stat = DSGetJobInfo(Hjob2, DSJ.JOBSTATUS)
If J2stat = DSJS.RUNFAILED
    Then Call DSLogFatal("Job DailyJob2 failed","JobControl")
End

* Now get a handle for the third job
Hjob3 = DSAttachJob("DailyJob3",DSJ.ERRFATAL)
* and run it
Dummy = DSRunJob(Hjob3,DSJ.RUNNORMAL)
* then wait for it to finish
Dummy = DSWaitForJob(Hjob3)

* Finally, get the finishing status for the third job and test it
J3stat = DSGetJobInfo(Hjob3, DSJ.JOBSTATUS)
If J3stat = DSJS.RUNFAILED
    Then Call DSLogFatal("Job DailyJob3 failed","JobControl")
End

```

Possible status conditions returned for a job are as follows.

A job that is in progress is identified by:

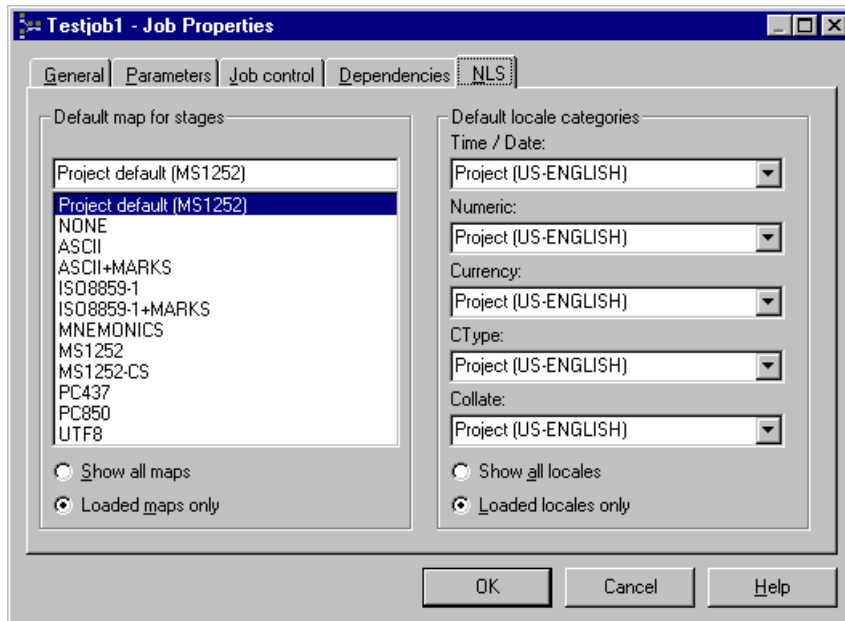
- DSJS.RUNNING – Job running; this is the only status that means the job is actually running.

Jobs that are not running may have the following statuses:

- DSJS.RUNOK – Job finished a normal run with no warnings.
- DSJS.RUNWARN – Job finished a normal run with warnings.
- DSJS.RUNFAILED – Job finished a normal run with a fatal error.
- DSJS.VALOK – Job finished a validation run with no warnings.
- DSJS.VALWARN – Job finished a validation run with warnings.
- DSJS.VALFAILED – Job failed a validation run.
- DSJS.RESET – Job finished a reset run.
- DSJS.STOPPED – Job was stopped by operator intervention (cannot tell run type).

## Specifying Maps and Locales

You can ensure that DataStage uses the correct character set map and formatting rules for your job by specifying character set maps and locales on the NLS page of the Job Properties dialog box.



## Defining Character Set Maps

The character set map defines the character set DataStage uses for this job. You can select a specific character set map from the list or accept the default setting for the whole project.

**Note:** The list contains all character set maps that are loaded and ready for use. You can view other maps that are supplied with DataStage by clicking **Show all maps**, but these maps cannot be used unless they are loaded using the DataStage Administrator. For more information, see *DataStage Administrator's Guide*.

## Defining Data Formats with Locales

Different countries and territories have different formatting conventions for common data types such as times, dates, numbers, and currency. The set of conventions used in a particular place with a particular language is called a locale. For example, there is a Canadian-French locale whose conventions differ from the French-French locale.

DataStage recognizes the locales for many territories. A default locale is set for each project during installation. You can override the default for a particular job by selecting the locale you require for each category on the **NLS** page of the Job Properties dialog box:

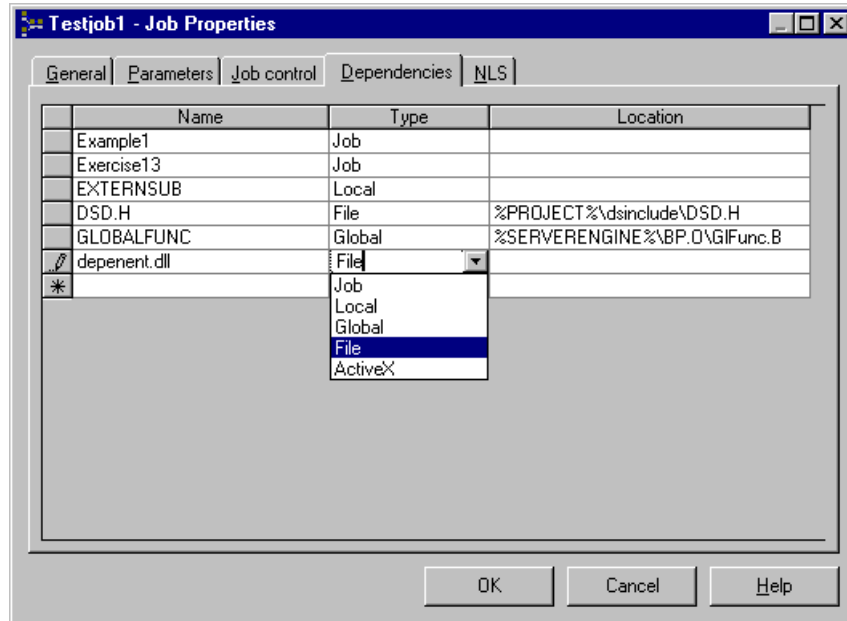
- **Time/Date** specifies the locale to use for formatting times and dates.
- **Numeric** specifies the locale to use for formatting numbers, for example, the thousands separator and radix character.
- **Currency** specifies the locale to use for monetary amounts, for example, the currency symbol and where it is placed.
- **CType** specifies the locale to use for determining character types, for example, which letters are uppercase and which lowercase.
- **Collate** specifies the locale to use for determining the order for sorted data.

In most cases you should use the same locale for every category to ensure that the data is formatted consistently.



## Specifying Job Dependencies

The **Dependencies** page of the Job Properties dialog box allows you to specify any dependencies the job has. These may be functions, routines, or other jobs that the job requires in order to run successfully.

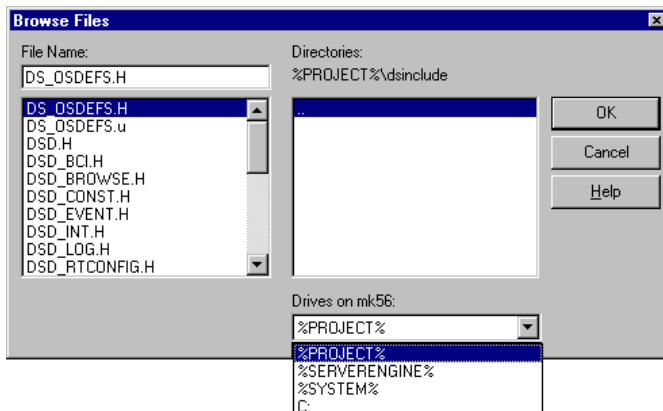


Enter details as follows:

- **Type.** The type of item upon which the job depends. Choose from the following:
  - **Job.** Released or unreleased job. If you have added a job on the **Job control** page (see page 5-38), this will automatically be included in the dependencies. If you subsequently delete the job from the job control routine, you must remove it from the dependencies list manually.
  - **Local.** Locally cataloged UniVerse BASIC functions and subroutines.
  - **Global.** Globally cataloged UniVerse BASIC functions and subroutines.
  - **File.** A standard file.
  - **ActiveX.** An ActiveX (OLE) object (not available on UNIX-based systems).

- **Name.** The name of the function or routine. The name required varies according to the **Type** of the dependency:
  - **Job.** The name of a released, or unreleased, job.
  - **Local.** The catalog name.
  - **Global.** The catalog name.
  - **File.** The file name.
  - **ActiveX.** The **Name** entry is actually irrelevant for ActiveX objects. Enter something meaningful to you (ActiveX objects are identified by the **Location** field).
- **Location.** The location of the dependency. A browse dialog is available to help with this. This location can be an absolute path, but it is recommended you specify a relative path using the following environment variables:
  - %SERVERENGINE% – UniVerse UV account directory (normally *C:\Ardent\DataStage\ServerEngine*).
  - %PROJECT% – Current project directory.
  - %SYSTEM% – System directory on Windows NT or */usr/lib* on UNIX.

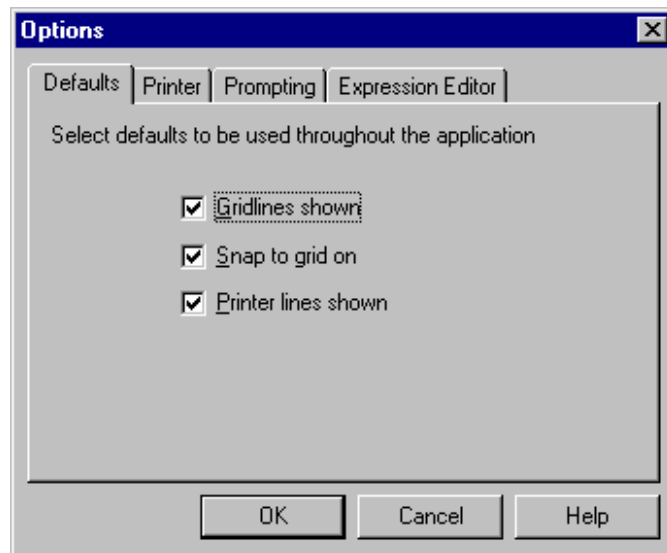
The Browse Files dialog box is as shown below. You cannot navigate to the parent directory of an environment variable.



When browsing for the location of a file on a UNIX server, there is an entry called **Root** in the **Base Locations** drop-down list.

## Specifying Designer Options

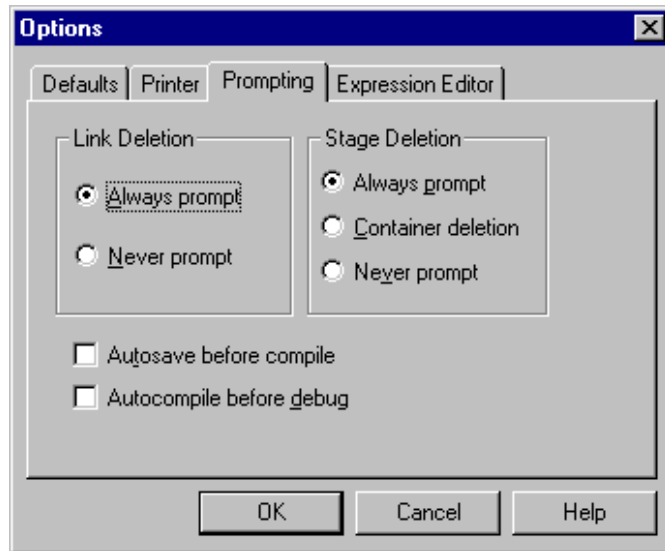
You can specify default display settings and the level of prompting used when the Designer is started. To specify the Designer options, choose **Tools ► Options...** . The Options dialog box appears:



This dialog box has four pages:

- **Defaults.** Determines the actions set in the Diagram window when you start the Designer. There are three check boxes on this page:
  - **Gridlines shown**
  - **Snap to grid on**
  - **Printer lines shown**These check boxes are selected by default. Clear the check box you do not want to set on startup.
- **Printer.** Specifies the printing orientation. When you choose **File ► Print**, the default printing orientation is determined by the setting on this page. You can choose portrait or landscape orientation. To use portrait, select the **Portrait orientation** check box. The default setting for this option is unchecked, i.e., landscape orientation is used.

- **Prompting.** Determines the level of prompting displayed when you delete a stage or link in a job design. This page has options for deleting a link and options for deleting a stage or a container.



The following options describe the action that occurs when you delete a link:

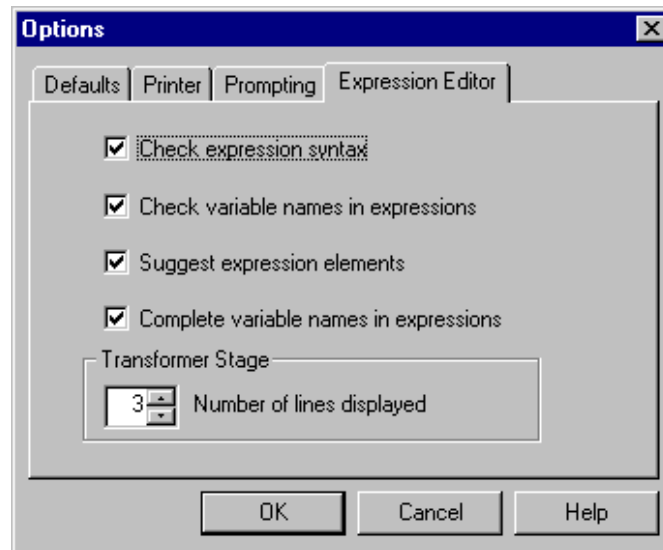
- **Always prompt.** You are always prompted to confirm the deletion.
- **Never prompt.** The chosen link or links are deleted without any confirmation. If you choose this option, containers (and the stages and links in them) are deleted without prompting.

The following options describe the action that occurs when you delete a stage or a container:

- **Always prompt.** You are always prompted to confirm the deletion.
- **Container deletion.** This is the default setting. You are prompted if you delete a container. If you do not choose this option or the **Always prompt** option, the container (and all the stages and links in it) is deleted.
- **Never prompt.** The chosen stage or container is deleted without any confirmation.

The **Autosave before compile** check box specifies whether a job is automatically saved when you compile it (without prompting). This check box is cleared by default, that is, the job is not saved automatically.

- **Expression Editor.** Specifies the features available in the DataStage Expression Editor. For more details on how to use the Expression Editor, see “The DataStage Expression Editor” on page 11-12.



There are four check boxes on this page:

- **Check expression syntax**
- **Check variable names in expressions**
- **Suggest expression elements**
- **Complete variable names in expressions**

These check boxes are selected by default. The settings are stored in the Repository and are used when you edit any job on this client machine.

**Number of lines displayed** allows you to specify the size of the Expression Editor box.

Click **OK** to save any changes and to close the Options dialog box. The chosen settings take effect immediately.



# 6

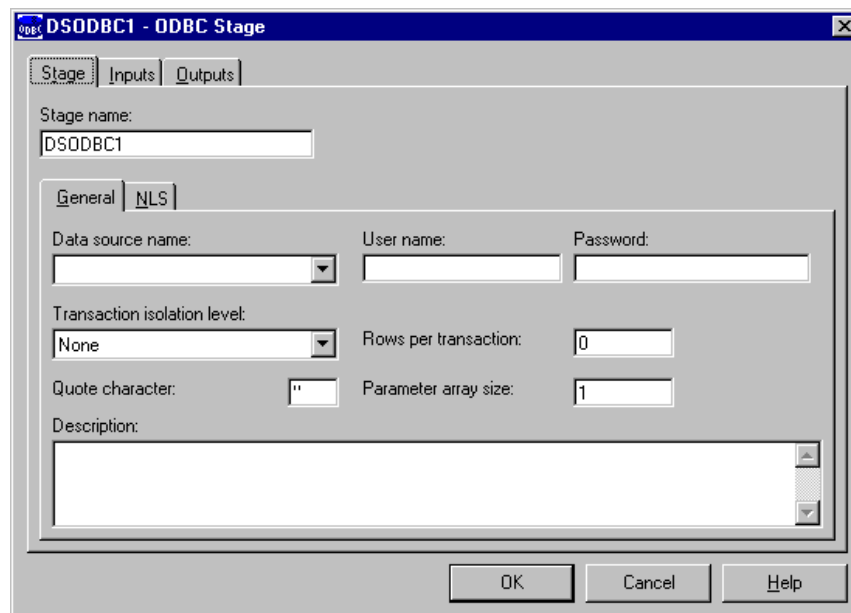
## ODBC Stages

ODBC stages are used to represent a database that supports the industry standard Open Database Connectivity API. You can use an ODBC stage to extract, write, or aggregate data.

Each ODBC stage can have any number of inputs or outputs. Input links specify the data you are writing. Output links specify the data you are extracting and any aggregations required.

You can specify the data on an input or output link using an SQL statement constructed by DataStage, a user-defined query, or a stored procedure.

When you edit an ODBC stage, the ODBC Stage dialog box appears:



This dialog box can have up to three pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** Displays the name of the stage you are editing. You can enter text to describe the purpose of the stage in the **Description** field. The **General** page defines the data source name. The **NLS** page defines a character set map to use with the stage, if required. For details see “Defining Character Set Maps” on page 6-4.

**Note:** You cannot change the name of the stage from this dialog box. For details on how to change stage names, see “Renaming Stages” on page 5-14.

- **Inputs.** This page is displayed only if you have an input link to this stage. Specifies the SQL table or stored procedure to use and the associated column definitions for each data input link. This page also specifies how data is written and contains the SQL statement or call syntax used to write the data.
- **Outputs.** This page is displayed only if you have an output link to this stage. Specifies the SQL tables or stored procedure to use and the associated column definitions for each data output link. This page also contains the SQL SELECT statement or call syntax used to extract the data.

You must perform the following steps to edit an ODBC stage:

1. Define the connection.
2. Define the data on the input links.
3. Define the data on the output links.

These steps are performed in the ODBC Stage dialog box. Click **OK** to close this dialog box. Changes are saved when you save the job design.

## Defining the Connection

To connect to an ODBC data source, you must install and configure a suitable ODBC driver on your system. For more information, see *DataStage Administrator's Guide*.

### ODBC Connection Parameters

The ODBC connection parameters are set on the **General** page on the **Stage** page.



To connect to an ODBC data source:

1. Choose the data source name from the Data source name drop-down list box. This drop-down list box contains all the data sources defined under the **Table Definitions** ► **ODBC** branch in the Repository.

**Note:** If the data source name you want is not listed, enter the name in the **Data source name** field or define a table definition. For details on how to import or create a table definition, see Chapter 4, “Setting Up a Project.”

You can also enter a job parameter in this field. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

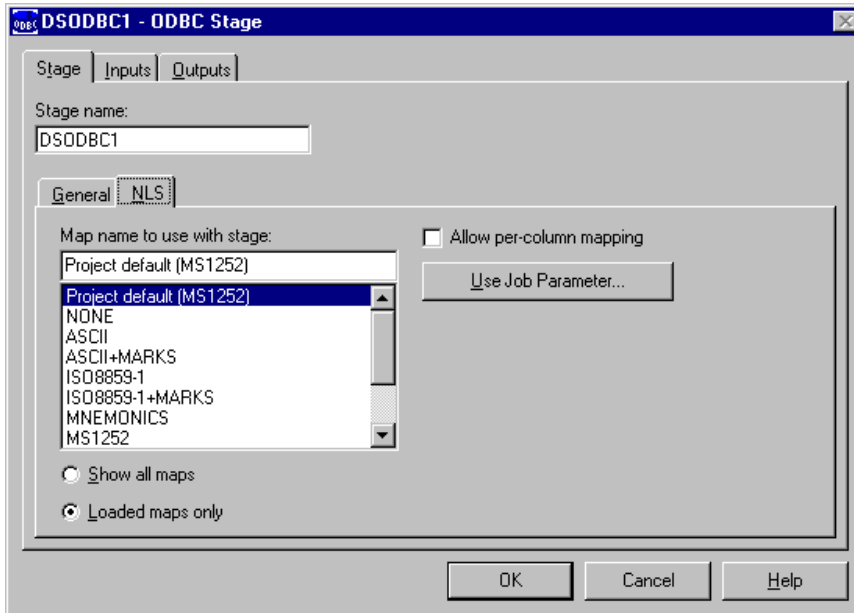
2. Enter the name to use to connect to the data source in the **User name** field. (You can also enter a job parameter in this field.)
3. Enter the password to use in the **Password** field. (You can also enter a job parameter in this field.)

**Note:** Certain ODBC drivers allow you to specify the user name and password to use on connection. If you are connecting to an ODBC data using a driver that has a user name and password already specified, you do not need to enter a user name and password on the **General** page.

4. Choose an appropriate transaction isolation level to use from the Transaction isolation level drop-down list box.
5. Enter a suitable value in the **Rows per transaction** field. This is the number of rows written before the data is committed to the data table. The default value is 0, that is, all the rows are written before being committed to the data table.
6. Enter a suitable value in the **Parameter array size** field. This is the number of rows written at a time. The default is 1, that is, each row is written in a separate operation.
7. Enter an optional description of the ODBC stage in the **Description** field.

## Defining Character Set Maps

You can define a character set map for an ODBC stage using the NLS page of the ODBC Stage dialog box.

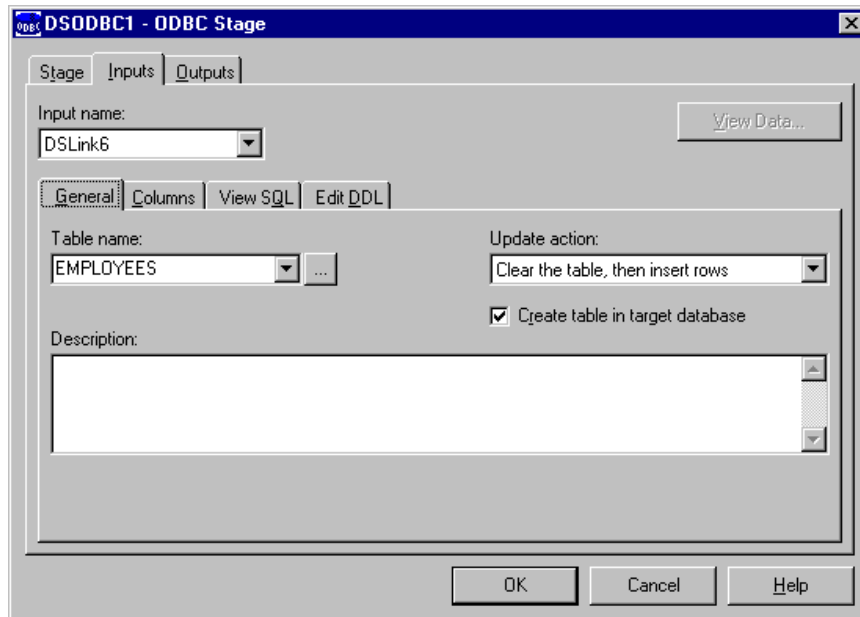


The default character set map (defined for the project or the job) can be changed by selecting a map name from the list. The page also has the following fields:

- **Show all maps.** Displays in the list all the maps supplied with DataStage. Maps cannot be used unless they have been loaded using the DataStage Administrator.
- **Loaded maps only.** Displays the maps that are loaded and ready for use.
- **Use Job Parameter...** Allows you to specify a character set map as a parameter to the job containing the stage. If the parameter has not yet been defined, you are prompted to define it from the Job Properties dialog box (see page 5-30).
- **Allow per-column mapping.** Allows character set maps to be specified for individual columns within the table definition. If per-column mapping is selected, an extra property, **NLS Map**, appears in the grid in the **Columns** page.

## Defining ODBC Input Data

When you write data to a table (or a stored procedure) in an ODBC database, the ODBC stage has an input link. The properties of this link and the column definitions of the data are defined on the **Inputs** page of the ODBC Stage dialog box.



The **Inputs** page has the following field and up to five pages:

- **Input name.** The name of the input link. Choose the link you want to edit from the Input name drop-down list box. This list box displays all the input links to the ODBC stage.
- **General.** This page is displayed by default. It contains the following parameters:
  - **Table name.** This field appears when the update action is *not Call stored procedure* or *User-defined SQL*. It is the name of the table the data is written to. Choose the table from the Table name drop-down list box. This list box contains all the tables defined under the **Table Definitions** > **ODBC** > **Data source** branch in the Repository. *Data source* is the data source name chosen on the **General** page on the **Stage** page.

**Note:** If the table you want is not listed, you need to define a table definition. For details on how to import or create a table definition, see “Table Definitions” on page 4-11. Alternatively, use **Browse...** to display the Table Definitions window and choose a suitable table definition.

You can also enter a job parameter in this field. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

- **Procedure name.** This field appears only when the update action is **Call stored procedure**. It is the name of the procedure the data is written to. Choose the stored procedure you want to use from the drop-down list box. This list box contains all the stored procedures defined under the **Table Definitions ► StoredProcedures** branch in the Repository for the specified DSN.

**Note:** If the stored procedure you want is not listed, you need to define it. For details on how to import or create a stored procedure definition, see “Stored Procedure Definitions” on page 4-35. Alternatively, use **Browse...** to search for the stored procedure you want.

- **Update action.** Specifies how the data is written. Choose the option you want from the drop-down list box:

**Clear the table, then insert rows.** Deletes the contents of the table and adds the new rows.

**Insert rows without clearing.** Inserts the new rows in the table.

**Replace existing rows completely.** Deletes the existing rows, then adds the new rows to the table.

**Update existing rows only.** Updates the existing data rows. Any rows in the data that do not exist in the table are ignored.

**Insert new or update existing rows.** New rows are added or, if the insert fails, the existing rows are updated.

**Update existing or insert new rows.** The existing data rows are updated or, if this fails, new rows are added.

**Call stored procedure.** Writes the data using a stored procedure. When you select this option, the **Procedure name** field appears.

**User-defined SQL.** Writes the data using a user-defined SQL statement. When you select this option, the **View SQL** page is replaced by the **Enter SQL** page.

- **Create table in target database.** Select this check box if you want to automatically create a table in the target database at run time. A table is created based on the defined column set for this stage. If you select this option, an additional page, **Edit DDL**, appears. This shows the SQL CREATE statement to be used for table generation.
- **Description.** Contains an optional description of the input link.
- **Columns.** This page contains the column definitions for the data written to the table or file. If you are using a stored procedure, the column definitions represent the stored procedure input parameters. You must have at least the same number of column definitions as expected parameters. The column definitions are used in the order they appear in the Columns grid. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.
- **View SQL.** This page displays the SQL statement or stored procedure call syntax used to write the data. You cannot edit this statement, but you can use **Copy** to copy it to the Clipboard for use elsewhere.
- **Enter SQL.** This page displays the user-defined SQL statement. It appears only when you set the update action to **User-defined SQL**.
- **Edit DDL.** This page appears if you have chosen to automatically generate a table at run time by selecting the **Create table in target database** check box on the **General** page. It displays the SQL CREATE statement that will be used to create the table. To generate the statement, click the **Create DDL** button. DataStage will connect to the target database and generate the statement. (If you are creating a table in a Sybase database, the Sybase database needs to have the “Data definition language in transaction” option set.) You can edit the statement on this page to make any required changes. This page also allows you to specify that any existing table by this name should be dropped first. If you do not select this option, and such a table already exists in the target database, then the create will fail.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the input link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

## Using a Generated Query

You can write data to an SQL table using an SQL statement constructed by DataStage. When you specify the table and the column definitions to use, the SQL statement is automatically constructed and can be viewed on the **View SQL** page.

To use a generated query:

1. Choose a table from the Table name drop-down list box on the **General** page.
2. Specify how you want the data to be written by choosing a suitable option from the Update action drop-down list box. There are six options for a generated query:
  - **Clear the table, then insert rows**
  - **Insert rows without clearing**
  - **Replace existing rows completely**
  - **Update existing rows only**
  - **Insert new or update existing rows**
  - **Update existing or insert new rows**

See page 6-6 for a description of each update action.

3. Enter an optional description of the input link in the **Description** field.
4. Click the **Columns** tab. The **Columns** page appears at the front of the **Inputs** page.
5. Edit the Columns grid to specify column definitions for the columns you want to write. For more information, see “Specifying Column Definitions” on page 5-18.

The SQL statement is automatically constructed using your chosen update action and the columns you have specified. You can now optionally view this SQL statement.

6. Click the **View SQL** tab. The **View SQL** page appears at the front of the **Inputs** page.
7. Click **OK** to close the ODBC Stage dialog box. Changes are saved when you save your job design.

## Using a User-Defined SQL Statement

Instead of writing data using an SQL statement constructed by DataStage, you can enter your own SQL statement for each ODBC input link.

To enter an SQL statement:

1. Choose **User-defined SQL** from the Update action drop-down list box. The **View SQL** page is replaced with the **Enter SQL** page.
2. Click the **Columns** tab. The **Columns** page appears at the front of the **Inputs** page.
3. Edit the Columns grid to specify column definitions for the columns you want to write. For more information, see “Specifying Column Definitions” on page 5-18.
4. Click the **Enter SQL** tab. The **Enter SQL** page appears at the front of the **Inputs** page.
5. Enter the SQL statement you want to use. This statement must contain the table name, the type of update action you want to perform, and the columns you want to write.

**Note:** You must also ensure that the statement contains the correct number of ? parameter markers. You must have a parameter marker for each column you have defined on the **Columns** page.

6. Click **OK** to close the ODBC Stage dialog box. Changes are saved when you save your job design.

## Using a Stored Procedure

Instead of writing data to a table using an SQL statement, you can write data to a stored procedure. The columns you define are bound (in order) to the input parameters in the stored procedure. The call syntax used to write the data is constructed by DataStage and can be viewed on the **View SQL** page.

The procedure is called once for each row of data presented to the input link.

To use a stored procedure:

1. Choose **Call stored procedure** from the Update action drop-down list box on the **General** page. The Table name drop-down list box is replaced by the Procedure name drop-down list box.
2. Choose the stored procedure from the Procedure name drop-down list box.
3. Enter an optional description of the input link in the **Description** field.
4. Click the **Columns** tab. The **Columns** page appears at the front of the **Inputs** page.

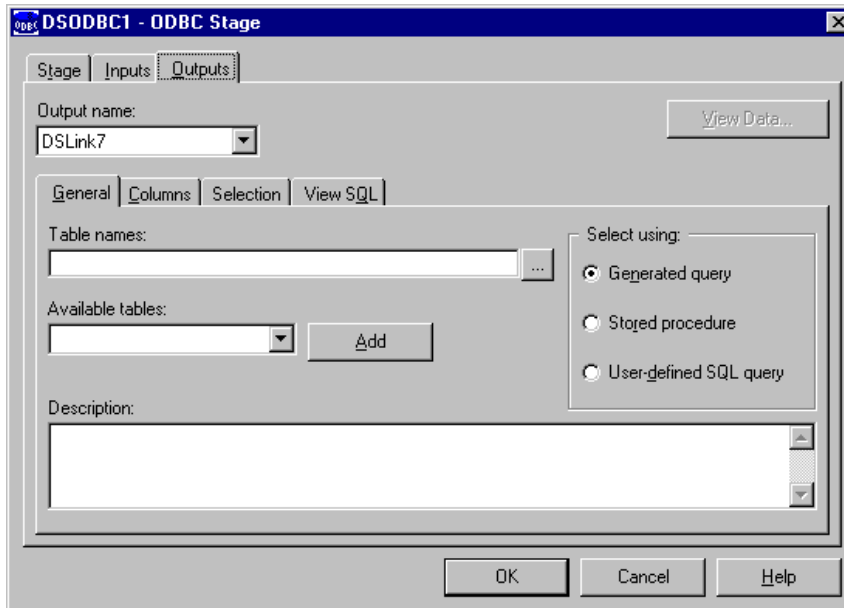
5. Edit the Columns grid to specify column definitions. For more information, see “Specifying Column Definitions” on page 5-18. The column definitions are used as the input parameters to the stored procedure. You must have at least the same number of column definitions as the number of expected input parameters.

The call statement is automatically constructed using the stored procedure name and the columns you have specified. You can now optionally view this SQL statement.

6. Click the **View SQL** tab. The **View SQL** page appears at the front of the **Inputs** page.
7. Click **OK** to close the ODBC Stage dialog box. Changes are saved when you save your job design.

## Defining ODBC Output Data

When you extract data from an ODBC data source, the ODBC stage has an output link. The properties of this link and the column definitions of the data are defined on the **Outputs** page in the ODBC Stage dialog box.





The **Outputs** page has the following field and up to six pages. The pages displayed depend on how you choose to specify the SQL statement to output the data.

- **Output name.** The name of the output link. Choose the link you want to edit from the Output name drop-down list box. This list box displays all the output links from the ODBC stage.
- **General.** Displayed by default. Contains the following features:
  - **Table names.** This field appears only when you select **Generated query** or **User-defined SQL query**. It contains the names of the tables or files being accessed. You can also use a job parameter to specify the table name. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.
  - **Available tables.** This drop-down list box appears only when you select **Generated query** or **User-defined SQL query**. It displays the names of the available tables or files that have definitions in the Repository.
  - **Add.** This button appears only when you select **Generated query** or **User-defined SQL query**. It adds a table from the Available tables drop-down list box to the **Table names** field.
  - **Stored procedure name.** This drop-down list box is available only when you select **Stored procedure**. It displays the name of the stored procedure you want to use. This drop-down list box displays all the stored procedure definitions under the **Table Definitions > StoredProcedures > DSN** branch in the Repository.
  - **Apply.** This button appears only when you select **Stored procedure**. It updates the **Columns** and **Parameters** pages with the settings for the chosen stored procedure.
  - **Generated query.** This is the default setting. When this option is selected, the **Selection** and **View SQL** pages appear. It specifies that the data is extracted using an SQL statement constructed by DataStage.
  - **Stored procedure.** Specifies that the data is extracted using a stored procedure. When this option is selected, the **View SQL** and **Parameters** pages appear.
  - **User-defined SQL query.** Specifies that the data is extracted using a user-defined SQL query. When this option is selected, the **SQL Query** page appears.
  - **Description.** Contains an optional description of the output link.

- **Browse...** . Displays the Table Definitions window, allowing you to choose a suitable table or stored procedure definition.
- **Columns.** Contains the column definitions for the data being output on the chosen link. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18. Also specifies which columns are aggregated.
- **Selection.** This page appears when you select **Generated query**. It contains optional SQL SELECT clauses for the conditional extraction of data.
- **View SQL.** This page appears when you select **Generated query** or **Stored procedure**. It displays the SQL statement used to extract the data from the chosen table or tables. The SQL statement exists in two forms and you can choose which one to display:
  - **SQL for reference inputs.** Choose this to view the SQL statement used when this link is a reference input to a Transformer stage.
  - **SQL for primary inputs.** Choose this to view the SQL statement used in all other cases.

You cannot edit the SQL statement, but you can use **Copy** to copy it to the Clipboard for use elsewhere.

- **Parameters.** Contains the input parameters for a chosen stored procedure. This page appears when you select **Stored procedure**.
- **SQL Query.** Contains a user-defined SQL query. This page appears when you select **User-defined SQL query**. This page is divided into two areas:
  - **SQL for primary inputs.** Contains a user-defined SQL query for a link that is a primary input to a Transformer stage, or an input to any other type of stage.
  - **SQL for reference inputs.** Contains a user-defined SQL query for a link that is a reference input to a Transformer stage.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the output link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

## Key Fields

The column definitions for output links contain a key field. Key fields are used to join primary and reference inputs to a Transformer stage. For details on how key fields are specified and used, see “Defining Input Column Key Expressions” on page 11-9.

## Using a Generated Query

When you select **Generated query**, data is extracted from an ODBC data source using an SQL SELECT statement constructed by DataStage. SQL SELECT statements have the following syntax:

```
SELECT clause FROM clause
      [WHERE clause]
      [GROUP BY clause]
      [HAVING clause]
      [ORDER BY clause];
```

When you specify the tables to use and the columns to be output from the ODBC stage, the SQL SELECT statement is automatically constructed and can be viewed by clicking the **View SQL** tab on the **Outputs** page.

**Note:** The **View SQL** page appears only when you select **Generated query** or **Stored procedure** from the **General** page on the **Outputs** page.

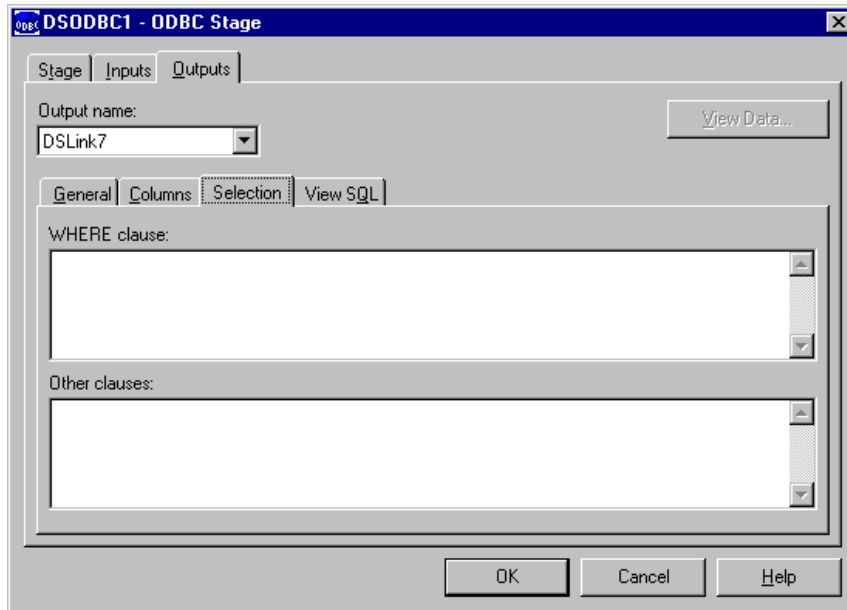
For example, if you extract the columns **Name**, **Address**, and **Phone** from a table called Table1, the SQL statement displayed on the **View SQL** page is:

```
SELECT Name, Address, Phone FROM Table1;
```

The SELECT and FROM clauses are the minimum required and are automatically generated by DataStage. However, you can use any of these SQL SELECT clauses:

SELECT clause	Specifies the columns to select from the database.
FROM clause	Specifies the tables containing the selected columns.
WHERE clause	Specifies the criteria that rows must meet to be selected.
GROUP BY clause	Groups rows to summarize results. See “Aggregating Data” on page 6-16 for a description of how this clause is used.
HAVING clause	Specifies the criteria that grouped rows must meet to be selected. See “Aggregating Data” on page 6-16 for a description of how this clause is used.
ORDER BY clause	Sorts selected rows.

If you want to use the additional SQL SELECT clauses, you must enter them on the **Selection** page on the **Outputs** page:



The **Selection** page is divided into two parts:

- **WHERE clause.** This text box allows you to insert an SQL WHERE clause to specify criteria that the data must meet before being selected.
- **Other clauses.** This text box allows you to insert a HAVING or an ORDER BY clause.

### Using a WHERE Clause

You can use a WHERE clause to:

- Select only the data that meets certain criteria
- Join two tables from the same data source

To use a WHERE clause, type the column and the condition into the **WHERE clause** text entry box.

For example, if you have a table (Sales1) containing sales data, you can choose to only output data where the value in the **Price** column is greater than \$10.00. In this case, enter:

```
Price>10
```

Alternatively, if you are extracting data from two tables in the data source, you can use a WHERE clause to relate a column in one table to a column in the another table.

For example, Table1 contains the columns **Pcode**, **OrderNo**, and **SaleDate** and Table2 contains **Pcode**, **CustNo**, **Quantity**, and **Cost**. You can use the WHERE clause to join the two tables together by the related column. In this case, the column is **Pcode** and you enter:

```
Table1.Pcode = Table2.Pcode
```

**Note:** Only one column definition called **Pcode** is loaded or inserted into the grid on the **Columns** page.

You can also use a job parameter in the WHERE clause. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

The SQL SELECT statement is automatically updated to include the WHERE clause. Click the **View SQL** tab to display the statement.

### Using a HAVING Clause

If you use an ODBC stage to aggregate data, you can use a HAVING clause to specify conditions the grouped data must meet before it is selected. For more information about using an ODBC stage to aggregate data, see “Aggregating Data” on page 6-16.

To use a HAVING clause, enter the clause, column, and condition into the **Other clauses** text entry box on the **Selection** page on the **Outputs** page.

For example, you could choose to only output summed quantities that are greater than or equal to 1000. In this case you enter:

```
HAVING SUM(QtySold)>=1000
```

The SQL SELECT statement is updated automatically. Click the **View SQL** tab to display the statement.

You can also use a job parameter in the HAVING clause. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

### Using an ORDER BY Clause

You can sort data based on a chosen column by including an ORDER BY clause in the SELECT statement. Records are sorted by data in the chosen column before being output. You can specify a column name or a column position and whether to sort in ascending or descending order.

To use an ORDER BY clause, enter the clause, column, and condition into the **Other clauses** text entry box on the **Selection** page on the **Outputs** page.

For example, if your table contains a **Name** column, you may want to sort the column alphabetically (A to Z). In this case you enter:

```
ORDER BY Name ASC
```

The SQL SELECT statement is updated automatically. Click the **View SQL** tab to display the statement.

### Aggregating Data

If you are using a generated query, you can use an ODBC stage to aggregate data at the source instead of using an intermediate Aggregator stage. By aggregating data you can add values in a particular column for all data records in a table. This summed total is then output from the stage.

You can aggregate data in two ways:

- Using an Aggregator stage. For more information, see Chapter 12, “Aggregator Stages.”
- Using an ODBC stage.

If you aggregate data using an ODBC stage, the columns to group by and sum together are also specified by the SQL SELECT statement. To specify the columns to group by and summarize, you must edit the column definitions in the Columns grid on the **Columns** page. For more information about editing column definitions, see “Specifying Column Definitions” on page 5-18.

For example, if you have a sales database (Sales1) it may contain the following columns: **Product**, **SaleDate**, and **QtySold**. If this database is updated daily, you have a record of how many of each product are sold each day. However, if you want to know how many of each product were sold since 01/01/96 you need to specify a WHERE clause for the **SaleDate** and group (and summarize) the data.

Because you want the total for each product, you need to group all the occurrences of the same value in the **Product** column and sum the value in the **QtySold** column.

To group by a column, click in the **Group** cell for the column definition you want to group by and choose **Yes** from the drop-down list. In the example, you would choose the **Product** column to edit.

To summarize a column, edit the **Derivation** cell for the column you want to aggregate (using SUM or COUNT). The **Derivation** cell contains by default the name of the table and column in the format *tablename.columnname*. You can edit this cell to add SUM or COUNT. In the example, you would edit the **Derivation** cell for the **QtySold** column. The resulting expression would be SUM(Sales1.QtySold).

When you group by or summarize columns, the SQL statement is automatically updated to include the GROUP BY clause and the aggregation expression. To view the SQL statement, click the **View SQL** tab on the **Outputs** page.

For example, the SQL statement for the example would be:

```
SELECT Product, SUM(QtySold) FROM Sales1
WHERE Saledate>=01/01/96
GROUP BY Product;
```

See “Using a WHERE Clause” on page 6-14 for details of how to use a WHERE clause.

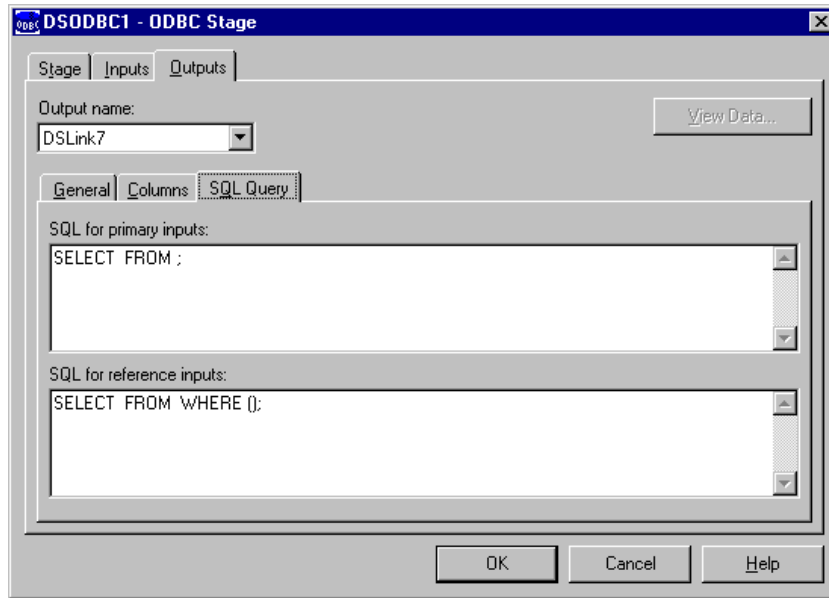
## Using a User-Defined SQL Statement

Instead of using the SQL statement constructed by DataStage, you can enter your own SQL statement for each ODBC output link.

To enter an SQL statement:

1. Click the **User-defined SQL query** option button on the **General** page on the **Outputs** page. The **SQL Query** tab appears.

2. Click the **SQL Query** tab. The **SQL Query** page appears at the front of the **Outputs** page:



When you first view this page, the **SQL for primary inputs** and **SQL for reference inputs** fields may contain the SQL statements constructed by DataStage. These are displayed if you selected **Generated query** or **Stored procedure** before selecting **User-defined SQL query**. You can modify or overwrite each statement to construct your own SQL query or call to a stored procedure.

The entries in these fields depend on whether the output is a primary input to a stage or a reference input to a Transformer stage:

- If the output is a primary input to any stage, whether or not it is a Transformer stage, edit the **SQL for primary inputs** field. The SQL query must contain the same number of columns (and column names) as the SQL statement constructed by DataStage.

You must ensure that the table definitions for the output link are correct and represent the columns that are expected. The result set generated from this statement returns at least one row.

**Note:** If more than one result set is produced, only the first set is used.



- If the output is a reference input to a Transformer stage, edit the **SQL for reference inputs** field. The SQL query must contain the same number of columns as the SQL statement constructed by DataStage. You must ensure that the table definitions for the output link are correct and represent the columns that are expected. The statement must have the same number of parameter values (?) as key columns on the link. The result set generated by this statement or procedure contains at most one row.
3. Click **OK** to close the ODBC Stage dialog box. Changes are saved when you save your job design.

## Using a Stored Procedure

Instead of using a user-defined SQL statement or one constructed by DataStage, you can use a stored procedure to define the data you want to extract for each ODBC output link.

The way in which you use a stored procedure depends on whether the output link is a primary input to another stage or a reference input to a Transformer stage.

**Note:** If the output from a stored procedure is used as a reference input to a Transformer stage, you may experience some loss of performance when the job is run.

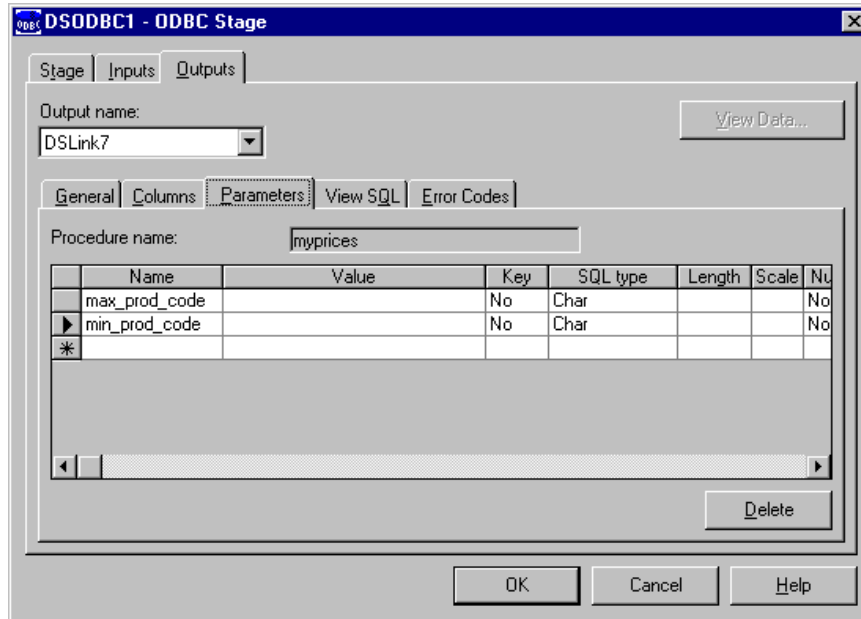
### Using a Stored Procedure for a Primary Input

If the ODBC output is a primary input to another stage in the job design, you must specify values for the stored procedure's parameters. To use a stored procedure for a primary input:

1. Click the **Stored procedure** option button on the **General** page on the **Outputs** page. The **View SQL** and **Parameters** tabs appear.
2. Choose the stored procedure you want to use from the Stored procedure name drop-down list box on the **General** page. This list box contains the names of the defined stored procedures under the **Table Definitions > StoredProcedures > DSN** branch in the Repository.

**Note:** If you can't see the name of the stored procedure you want to use, you must define it using the DataStage Manager. For more information about importing, creating, and editing stored procedure definitions, see "Stored Procedure Definitions" on page 4-35. Alternatively, use **Browse...** to search the system for the stored procedure you want.

3. Click the **Apply** button. The **Columns** and **Parameters** pages are updated with the column and parameter definitions for the chosen stored procedure.
4. Click the **Parameters** tab. The **Parameters** page appears at the front of the **Outputs** page:



5. Click in the **Value** cell for each parameter and enter suitable values. You can enter constants or use job parameters. For more information about using job parameters, see “Specifying Job Parameters” on page 5-33.

The call syntax used to extract the data is automatically updated with the parameter values. You can view this syntax on the **View SQL** page.

6. Click **OK** to close the ODBC Stage dialog box. Changes are saved when you save your job design.

When the job runs, the stored procedure is called once with the given parameter values. The result set generated should contain at least one row.

## Using a Stored Procedure for a Reference Input

If the ODBC output is a reference input to a Transformer stage, you must check that there is a parameter definition for each key column on the output link. When the job is run, the parameter values are set to the values in the corresponding key columns. To use a stored procedure for a reference input:

1. Click the **Stored procedure** option button on the **General** page on the **Outputs** page. The **View SQL** and **Parameters** tabs appear.
2. Choose the stored procedure you want to use from the Stored procedure name drop-down list box. This list box contains the names of the defined stored procedures under the **Table Definitions > StoredProcedures > DSN** branch in the Repository.

**Note:** If you can't see the name of the stored procedure you want to use, you must define it using the DataStage Manager. For more information about importing, creating, and editing stored procedure definitions, see "Stored Procedure Definitions" on page 4-35. Alternatively, use **Browse...** to search the system for the stored procedure you want.

3. Click the **Apply** button. The **Columns** and **Parameters** pages are updated with the column and parameter definitions for the chosen stored procedure.
4. Click the **Parameters** tab. The **Parameters** page appears at the front of the **Outputs** page.
5. Check that there is a parameter definition for each key column specified on the **Columns** page.

The call syntax used to extract the data is automatically updated. You can view this syntax on the **View SQL** page.

6. Click **OK** to close the ODBC Stage dialog box. Changes are saved when you save your job design.

When the job runs, the stored procedure is called once for each row on the output link. The values in the key columns on the **Columns** page are bound to the stored procedure parameters. The result set generated should contain at least one row. The columns in the result set should match the column definitions on the output link.



# 7

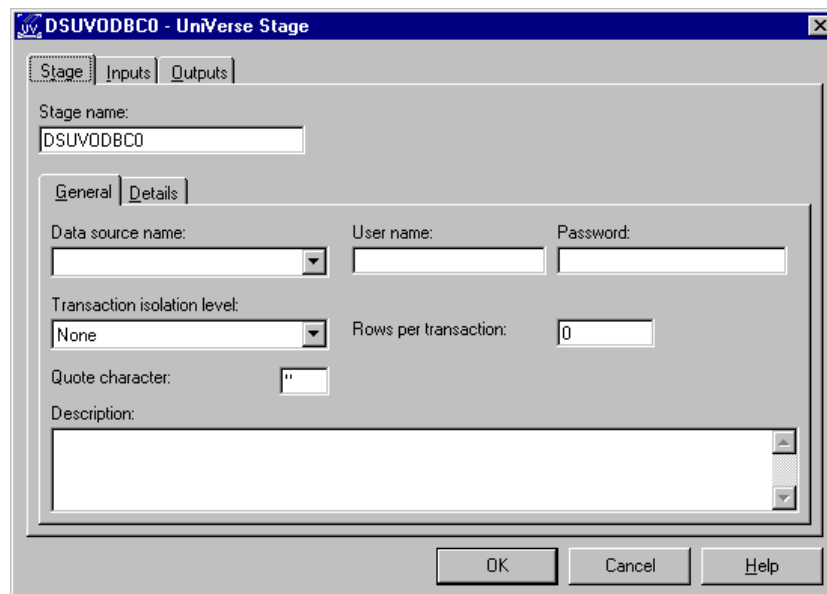
## UniVerse Stages

UniVerse stages allow you to connect to UniVerse tables using SQL. You can use a UniVerse stage to extract, write, or aggregate data. (Use a Hashed File stage to connect to UniVerse files.)

Each UniVerse stage can have any number of inputs or outputs. Input links specify the data you are writing. Output links specify the data you are extracting and any aggregations required.

You can specify the data on an input or output link using an SQL statement constructed by DataStage or a user-defined query.

When you edit a UniVerse stage, the UniVerse Stage dialog box appears:



This dialog box can have up to three pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** Displays the name of the stage you are editing. You can enter text to describe the purpose of the stage in the **Description** field. The **General** page defines the data source name.  
**Note:** You cannot change the name of the stage from this dialog box. For details on how to change stage names, see “Renaming Stages” on page 5-14.
- **Inputs.** This page is displayed only if you have an input link to this stage. Specifies the UniVerse table to use and the associated column definitions for each data input link. This page also specifies how data is written and contains the SQL statement or call syntax used to write the data.
- **Outputs.** This page is displayed only if you have an output link to this stage. Specifies the UniVerse tables to use and the associated column definitions for each data output link. This page also contains the SQL SELECT statement or call syntax used to extract the data.

You must perform the following steps to edit a UniVerse stage:

1. Define the connection.
2. Define the data on the input links.
3. Define the data on the output links.

These steps are performed in the UniVerse Stage dialog box. Click **OK** to close this dialog box. Changes are saved when you save the job design.

## Defining the Connection

The UniVerse connection parameters are set on the **General** page on the **Stage** page. To connect to a UniVerse data source, the data source must be configured in the *uvodbc.config* file. For more information, see *DataStage Administrator's Guide*.

### UniVerse Connection Parameters

The UniVerse connection parameters are set on the **General** and **Details** pages on the **Stage** page. To connect to a UniVerse data source:

1. Choose the data source name from the Data source name drop-down list box. This drop-down list box contains all the data sources defined under the **Table Definitions** ► **UniVerse** branch in the Repository.

**Note:** If the data source name you want is not listed, you can either enter the name in the **Data source name** field or define a table definition. For details on how to import or create a table definition, see Chapter 4, “Setting Up a Project.”

You can also enter a job parameter in this field. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

2. Enter the user name to use in the **User name** field. You can enter a job parameter here, in the form *#parameter#*.
3. Enter the password to use in the **Password** field. You can enter a job parameter here, in the form *#parameter#*, but it will be displayed as asterisks.

**Note:** If you are connecting to a remote UniVerse system using LAN Manager, or accessing a data file in *localuv*, you do not need to enter a user name and password. In the case of *localuv* data sources, the **User name** and **Password** fields are disabled.

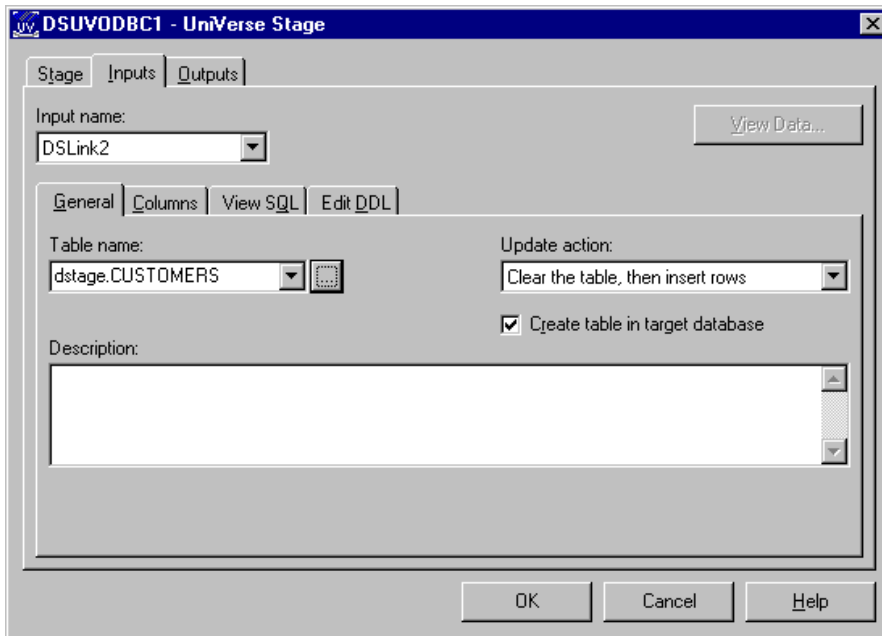
4. Choose an appropriate transaction isolation level to use from the Transaction isolation level drop-down list box.
5. Enter an optional description of the UniVerse stage in the **Description** field.
6. If you are using a DSN other than *localuv*, click the **Details** tab. The **Details** page appears at the front of the **Stage** page.
7. Specify where to find the file to use by clicking the appropriate option button:
  - **Use account name.** Enter the account name in the **Account name** field.
  - **Use directory path.** Enter the directory path where the file is stored in the **Directory path** field or use **Browse...** to search the system for a suitable directory. For more information about using **Browse...**, see “Browsing Server Directories” on page 5-23.

**Note:** If you are accessing a file on *localuv*, you do not need to specify an account or directory path, and the **Details** tab is disabled.

You can use a job parameter in any of the fields described in steps 1, 2, 3, 4, and 7. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

## Defining UniVerse Input Data

When you write data to a table in a UniVerse database, the UniVerse stage has an input link. The properties of this link and the column definitions of the data are defined on the **Inputs** page of the UniVerse Stage dialog box.



The **Inputs** page has the following field and up to five pages:

- **Input name.** The name of the input link. Choose the link you want to edit from the Input name drop-down list box. This list box displays all the input links to the UniVerse stage.
- **General.** This page is displayed by default. It contains the following parameters:
  - **Table name.** The name of the table or UniVerse file the data is written to. Choose the table or file from the Table name drop-down list box. This list box contains all the tables defined under the **Table Definitions** ► **UniVerse** ► **Data source** branch in the Repository. *Data source* is the data source name chosen on the **General** page on the **Stage** page.



**Note:** If the table you want is not listed, you need to define a table definition. For details on how to import or create a table definition, see “Table Definitions” on page 4-11. Alternatively, use **Browse...** to display the Table Definitions window and choose a suitable table definition.

You can also enter a job parameter in this field. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

This field appears when the update action is *not* **User-defined SQL**.

- **Update action.** Specifies how the data is written. Choose the option you want from the drop-down list box:

**Clear the table, then insert rows.** Deletes the contents of the table and adds the new rows.

**Insert rows without clearing.** Inserts the new rows in the table.

**Replace existing rows completely.** Deletes the existing rows, then adds the new rows to the table.

**Update existing rows only.** Updates the existing data rows. Any rows in the data that do not exist in the table are ignored.

**Update existing rows or insert new ones.** The existing data rows are updated and new rows are added.

**User-defined SQL.** The data is written using a user-defined SQL statement. When you select this option, the **View SQL** page is replaced by the **Enter SQL** page.

- **Create table in target database.** Select this check box if you want to automatically create a table in the target database at run time. A table is created based on the defined column set for this stage. If you select this box, an additional page, **Edit DDL**, appears. This shows the SQL CREATE statement to be used for table generation.
- **Description.** Contains an optional description of the input link.
- **Columns.** Contains the column definitions for the data written to the table or file. The column definitions are used in the order they appear in the Columns grid. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

- **View SQL.** Displays the SQL statement call syntax used to write the data. You cannot edit this statement, but you can use **Copy** to copy it to the Clipboard for use elsewhere. (This page is not displayed if the update action is **User-defined SQL**.)
- **Enter SQL.** Displays the user-defined SQL statement. This page appears only when you set the update action to **User-defined SQL**.
- **Edit DDL.** This page appears if you have chosen to automatically generate a table at run time by selecting the **Create table in target database** check box on the **General** page. It displays the SQL CREATE statement that will be used to create the table. To generate the statement, click the **Create DDL** button. DataStage will connect to the target database and generate the statement. You can edit the statement on this page to make any required changes. This page also allows you to specify that any existing table by this name should be dropped first. If you do not select this option, and such a table already exists in the target database, then the create will fail.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the input link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

## Using a Generated Query

You can write data to a UniVerse table using an SQL statement constructed by DataStage. When you specify the table and the column definitions to use, the SQL statement is automatically constructed and can be viewed on the **View SQL** page.

To use a generated query:

1. Choose a table from the Table name drop-down list box on the **General** page.
2. Specify how you want the data to be written by choosing a suitable option from the Update action drop-down list box. There are five options for a generated query:
  - **Clear the table, then insert rows**
  - **Insert rows without clearing**
  - **Replace existing rows completely**
  - **Update existing rows only**
  - **Update existing rows or insert new ones**

See page 7-5 for a description of each update action.

3. Enter an optional description of the input link in the **Description** field.

4. Click the **Columns** tab. The **Columns** page appears at the front of the **Inputs** page.
5. Edit the Columns grid to specify column definitions for the columns you want to write. For more information, see “Specifying Column Definitions” on page 5-18.  
The SQL statement is automatically constructed using your chosen update action and the columns you have specified. You can now optionally view this SQL statement.
6. Click the **View SQL** tab. The **View SQL** page appears at the front of the **Inputs** page.
7. Click **OK** to close the UniVerse Stage dialog box. Changes are saved when you save your job design.

## Using a User-Defined SQL Statement

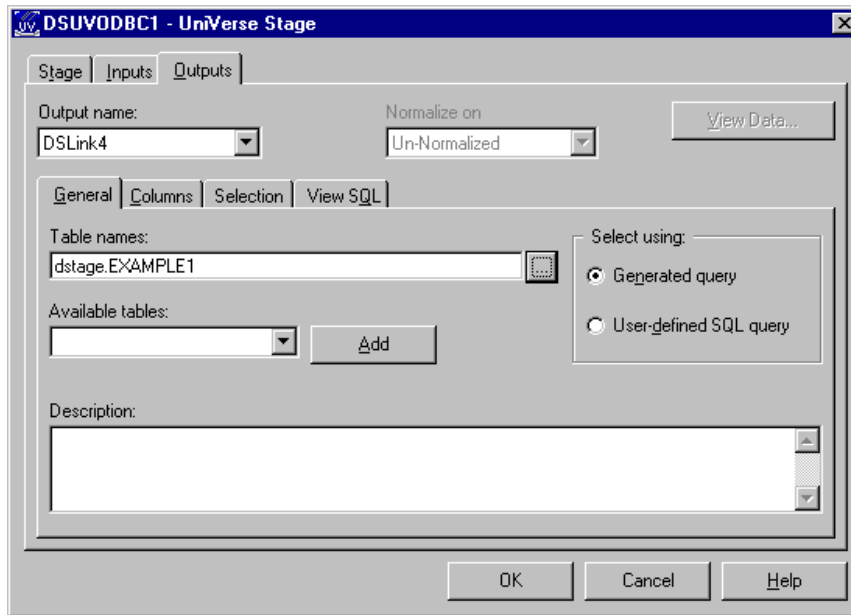
Instead of writing data using an SQL statement constructed by DataStage, you can enter your own SQL statement for each UniVerse input link.

To enter an SQL statement:

1. Choose **User-defined SQL** from the Update action drop-down list box on the **General** page. The **View SQL** tab is replaced with the **Enter SQL** tab.
2. Click the **Columns** tab. The **Columns** page appears at the front of the **Inputs** page.
3. Edit the Columns grid to specify column definitions for the columns you want to write. For more information, see “Specifying Column Definitions” on page 5-18.
4. Click the **Enter SQL** tab. The **Enter SQL** page appears at the front of the **Inputs** page.
5. Enter the SQL statement you want to use. This statement must contain the table name, the type of update action you want to perform, and the columns you want to write.  
**Note:** You must also ensure that the statement contains the correct number of ? parameter markers. You must have a parameter marker for each column you have defined on the **Columns** page.
6. Click **OK** to close the UniVerse Stage dialog box. Changes are saved when you save your job design.

## Defining UniVerse Output Data

When you extract data from a UniVerse data source, the UniVerse stage has an output link. The properties of this link and the column definitions of the data are defined on the **Outputs** page in the UniVerse Stage dialog box.



The **Outputs** page has the following two fields and up to five pages. The pages displayed depend on how you choose to specify the SQL statement to output the data.

- **Output name.** The name of the output link. Choose the link you want to edit from the Output name drop-down list box. This list box displays all the output links from the UniVerse stage.
- **Normalize on.** This drop-down list box allows you to normalize (or unnest) data. You can normalize either on an association or on a single unassociated multivalued column. The Normalize on drop-down list box is only enabled for nonreference output links where meta data has been defined that contains multivalued fields.

- **General.** Displayed by default. Contains the following features:
  - **Table names.** Contains the names of the tables or files being accessed in a comma-separated list. You can also use a job parameter to specify the table name. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.
  - **Available tables.** Displays the names of the available tables or files that have definitions in the Repository.
  - **Add.** Adds a table from the Available tables drop-down list box to the **Table names** field.
  - **Generated query.** Specifies that the data is extracted using an SQL statement constructed by DataStage. This is the default setting. When this option is selected, the **Selection** and **View SQL** pages appear.
  - **User-defined SQL query.** Specifies that the data is extracted using a user-defined SQL query. When this option is selected, the **SQL Query** page appears.
  - **Description.** Contains an optional description of the output link.
  - **Browse...** . Displays the Table Definitions window, allowing you to choose a suitable table or stored procedure definition.
- **Columns.** Contains the column definitions for the data being output on the chosen link. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18. Also specifies which columns are aggregated.
- **Selection.** Contains optional SQL SELECT clauses for the conditional extraction of data. This page appears when you select **Generated query**.
- **View SQL.** Displays the SQL statement used to extract the data from the chosen table or tables. This page appears when you select **Generated query**. The SQL statement exists in two forms and you can choose which one to display:
  - **SQL for reference inputs.** Choose this to view the SQL statement used when this link is a reference input to a Transformer stage.
  - **SQL for primary inputs.** Choose this to view the SQL statement used in all other cases.

You cannot edit the SQL statement, but you can use **Copy** to copy it to the Clipboard for use elsewhere.

- **SQL Query.** Contains a user-defined SQL query. This page appears when you select **User-defined SQL query**. This page is divided into two areas:
  - **SQL for primary inputs.** Contains a user-defined SQL query for a link that is a primary input to a Transformer stage, or an input to any other type of stage.
  - **SQL for reference inputs.** Contains a user-defined SQL query for a link that is a reference input to a Transformer stage.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the input link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

## Key Fields

The column definitions for output links contain a key field. Key fields are used to join primary and reference inputs to a Transformer stage. For details on how key fields are specified and used, see “Defining Input Column Key Expressions” on page 11-9.

## Using a Generated Query

When you select **Generated query**, data is extracted from a UniVerse data source using an SQL SELECT statement constructed by DataStage. SQL SELECT statements have the following syntax:

```
SELECT clause FROM clause
    [WHERE clause]
    [GROUP BY clause]
    [HAVING clause]
    [ORDER BY clause];
```

When you specify the tables to use and the columns to be output from the UniVerse stage, the SQL SELECT statement is automatically constructed and can be viewed by clicking the **View SQL** tab on the **Outputs** page.

**Note:** The **View SQL** page appears only when you select **Generated query** on the **General** page on the **Outputs** page.

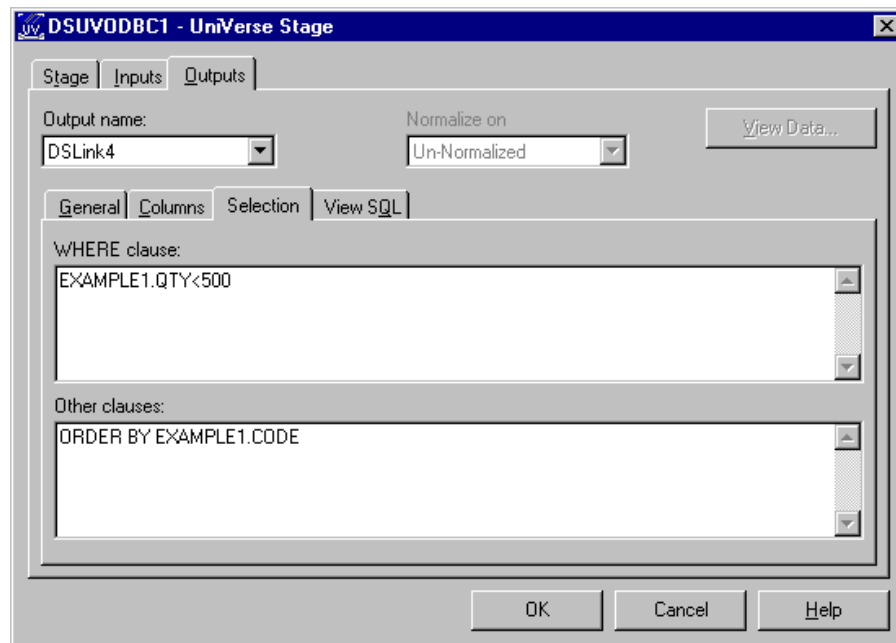
For example, if you extract the columns **Name**, **Address**, and **Phone** from a table called Table1, the SQL statement displayed on the **View SQL** page is:

```
SELECT Name, Address, Phone FROM Table1;
```

The SELECT and FROM clauses are the minimum required and are automatically generated by DataStage. However, you can use any of these SQL SELECT clauses:

- |                 |  |
|-----------------|--|
| SELECT clause   | Specifies the columns to select from the database.   |
| FROM clause     | Specifies the tables containing the selected columns.  |
| WHERE clause    | Specifies the criteria that rows must meet to be selected.   |
| GROUP BY clause | Groups rows to summarize results. See “Aggregating Data” on page 7-13 for a description of how this clause is used.                                  |
| HAVING clause   | Specifies the criteria that grouped rows must meet to be selected. See “Aggregating Data” on page 7-13 for a description of how this clause is used. |
| ORDER BY clause | Sorts selected rows.   |

If you want to use the additional SQL SELECT clauses, you must enter them on the **Selection** page on the **Outputs** page:



The **Selection** page is divided into two parts:

- **WHERE clause.** This text box allows you to insert an SQL WHERE clause to specify criteria that the data must meet before being selected.
- **Other clauses.** This text box allows you to insert a HAVING or an ORDER BY clause.

### Using a WHERE Clause

You can use a WHERE clause to:

- Select only the data that meets certain criteria
- Join two tables from the same data source

To use a WHERE clause, type the column and the condition into the **WHERE clause** text entry box.

For example, if you have a table (Sales1) containing sales data, you can choose to only output data where the value in the **Price** column is greater than \$10.00. In this case, enter:

```
Price>10
```

Alternatively, if you are extracting data from two tables in the data source, you can use a WHERE clause to relate a column in one table to a column in the another table.

For example, Table1 contains the columns **Pcode**, **OrderNo**, and **SaleDate** and Table2 contains **Pcode**, **CustNo**, **Quantity**, and **Cost**. You can use the WHERE clause to join the two tables together by the related column. In this case, the column is **Pcode** and you enter:

```
Table1.Pcode = Table2.Pcode
```

**Note:** Only one column definition called **Pcode** is loaded or inserted into the grid on the **Columns** page.

You can also use a job parameter in the WHERE clause. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

The SQL SELECT statement is automatically updated to include the WHERE clause. Click the **View SQL** tab to display the statement.



## Using a HAVING Clause

If you use a UniVerse stage to aggregate data, you can use a HAVING clause to specify conditions the grouped data must meet before it is selected. For more information about using a UniVerse stage to aggregate data, see “Aggregating Data” on page 7-13.

To use a HAVING clause, enter the clause, column, and condition into the **Other clauses** text entry box on the **Selection** page on the **Outputs** page.

For example, you could choose to only output summed quantities that are greater than or equal to 1000. In this case you enter:

```
HAVING SUM(QtySold)>=1000
```

The SQL SELECT statement is updated automatically. Click the **View SQL** tab to display the statement.

You can also use a job parameter in the HAVING clause. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.

## Using an ORDER BY Clause

You can sort data based on a chosen column by including an ORDER BY clause in the SELECT statement. Records are sorted by data in the chosen column before being output. You can specify a column name or a column position and whether to sort in ascending or descending order.

To use an ORDER BY clause, enter the clause, column, and condition into the **Other clauses** text entry box on the **Selection** page on the **Outputs** page.

For example, if your table contains a **Name** column, you may want to sort the column alphabetically (A to Z). In this case you enter:

```
ORDER BY Name ASC
```

The SQL SELECT statement is updated automatically. Click the **View SQL** tab to display the statement.

## Aggregating Data

If you are using a generated query, you can use a UniVerse stage to aggregate data at the source instead of using an intermediate Aggregator stage. By aggregating data you can add values in a particular column for all data records in a table. This summed total is then output from the stage.

You can aggregate data in two ways:

- Using an Aggregator stage. For more information, see Chapter 12, “Aggregator Stages.”
- Using a UniVerse stage.

If you aggregate data using a UniVerse stage, the columns to group by and sum together are also specified by the SQL SELECT statement. To specify the columns to group by and summarize, you must edit the column definitions in the Columns grid on the **Columns** page. For more information about editing column definitions, see “Specifying Column Definitions” on page 5-18.

For example, if you have a sales database (Sales1) it may contain the following columns: **Product**, **SaleDate**, and **QtySold**. If this database is updated daily, you have a record of how many of each product are sold each day. However, if you want to know how many of each product were sold since 01/01/96 you need to specify a WHERE clause for the **SaleDate** and group (and summarize) the data.

Because you want the total for each product, you need to group all the occurrences of the same value in the **Product** column and sum the value in the **QtySold** column.

To group by a column, click in the **Group** cell for the column definition you want to group by and choose **Yes** from the drop-down list. In the example, you would choose the **Product** column to edit.

To summarize a column, click in the **Derivation** cell for the column you want to aggregate (using SUM or COUNT). The **Derivation** cell contains by default the name of the table and column in the format *tablename.columnname*. You can edit this cell to add SUM or COUNT. In the example, you would edit the **Derivation** cell for the **QtySold** column. The resulting expression would be SUM(Sales1.QtySold).

When you group by or summarize columns, the SQL statement is automatically updated to include the GROUP BY clause and the aggregation expression. To view the SQL statement, click the **View SQL** tab on the **Outputs** page.

For example, the SQL statement for the example would be:

```
SELECT Product, SUM(QtySold) FROM Sales1
WHERE Saledate>=01/01/96
GROUP BY Product;
```

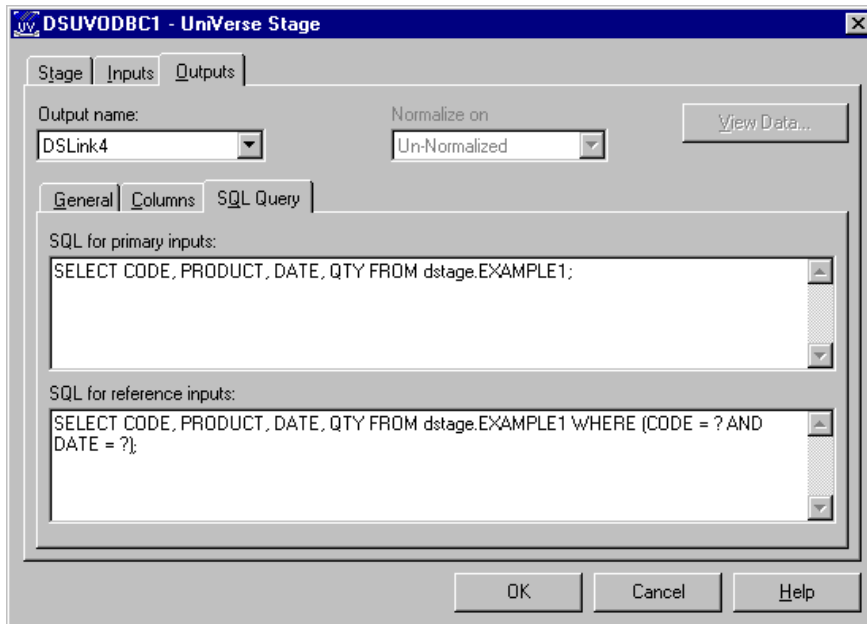
See “Using a WHERE Clause” on page 7-12 for details of how to use a WHERE clause.

## Using a User-Defined SQL Statement

Instead of using the SQL statement constructed by DataStage, you can enter your own SQL statement for each UniVerse output link.

To enter an SQL statement:

1. Click the **User-defined SQL query** option on the **General** page on the **Outputs** page. The **SQL Query** tab appears.
2. Click the **SQL Query** tab. The **SQL Query** page appears at the front of the **Outputs** page:



When you first view this page, the **SQL for primary inputs** and **SQL for reference inputs** fields may contain the SQL statements constructed by DataStage. These are displayed if you selected **Generated query** before selecting **User-defined SQL query**. You can modify or overwrite each statement to construct your own SQL query or call to a stored procedure.

The entries in these fields depend on whether the output is a primary input to a stage or a reference input to a Transformer stage:

- If the output is a primary input to any stage, whether or not it is a Transformer stage, edit the **SQL for primary inputs** field. The SQL query must contain the same number of columns (and column names) as the SQL statement constructed by DataStage.

You must ensure that the table definitions for the output link are correct and represent the columns that are expected. The result set generated from this statement returns at least one row.

**Note:** If more than one result set is produced, only the first set is used.

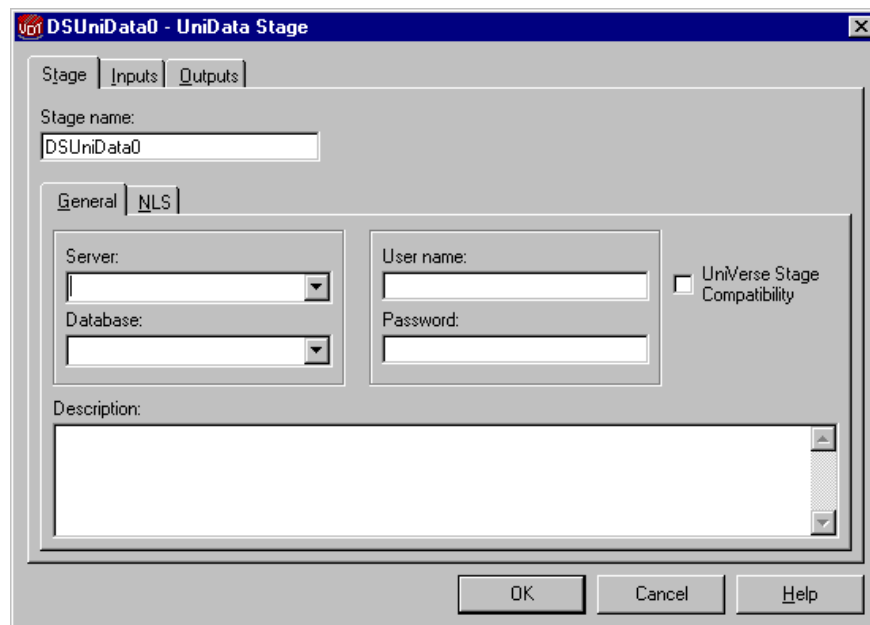
- If the output is a reference input to a Transformer stage, edit the **SQL for reference inputs** field. The SQL query must contain the same number of columns as the SQL statement constructed by DataStage. You must ensure that the table definitions for the output link are correct and represent the columns that are expected. The statement must have the same number of parameter values (?) as key columns on the link. The result set generated by this statement or procedure contains at most one row.
3. Click **OK** to close the UniVerse Stage dialog box. Changes are saved when you save your job design.

# 8

## UniData Stages

UniData stages represent a UniData file. You can use a UniData stage to extract or write data, or to act as an intermediate file in a job.

Each UniData stage can have any number of inputs or outputs. When you edit a UniData stage, the UniData Stage dialog box appears:



This dialog box can have up to three pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** Displays the name of the stage you are editing. This page has a **General** page, where you can enter text to describe the purpose of the stage in the **Description** field. You can also specify where the data files are by selecting a data source name and supplying logon details for the server where the data source is located.
    - **Server.** Choose a server name from the drop-down list or type a server name. The list is populated from the **Table Definitions ► UniData** branch in the Manager. You can use a job parameter in this field. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.
    - **Database.** Choose a database from the drop-down list or type a database name. The list is populated from the **Table Definitions ► UniData** branch in the Manager. You can use a job parameter in this field.
    - **User name.** Enter the user name to be used for logging on to the server. You can use a job parameter in this field.
    - **Password.** Enter the password to be used for logging on to the server. You can use a job parameter in this field.
    - **UniVerse Stage Compatibility.** Select this check box to ensure that any job conversions will work correctly. With this option selected, the date or time will be represented in ISO format (depending on the Extended type) and numerics will be scaled according to the meta data. (The job conversion utility is a special standalone tool – it is not available within the DataStage Designer.)
- The **NLS** page defines a character set map to use with this stage if NLS is enabled. For details, see “Defining Character Set Maps” on page 8-3.
- Note:** You cannot change the name of the stage from this dialog box. For details on how to change stage names, see “Renaming Stages” on page 5-14.
- **Inputs.** This page is displayed only if you have an input link to this stage. Specifies the data file to use and the associated column definitions for each data input link. This page also specifies how data is written to the data file.
  - **Outputs.** This page is displayed only if you have an output link to this stage. Specifies the data file to use and the associated column definitions for each data output link.

Click **OK** to close this dialog box. Changes are saved when you save the job.

If you encounter an error when connecting to a UniData data source, such as:

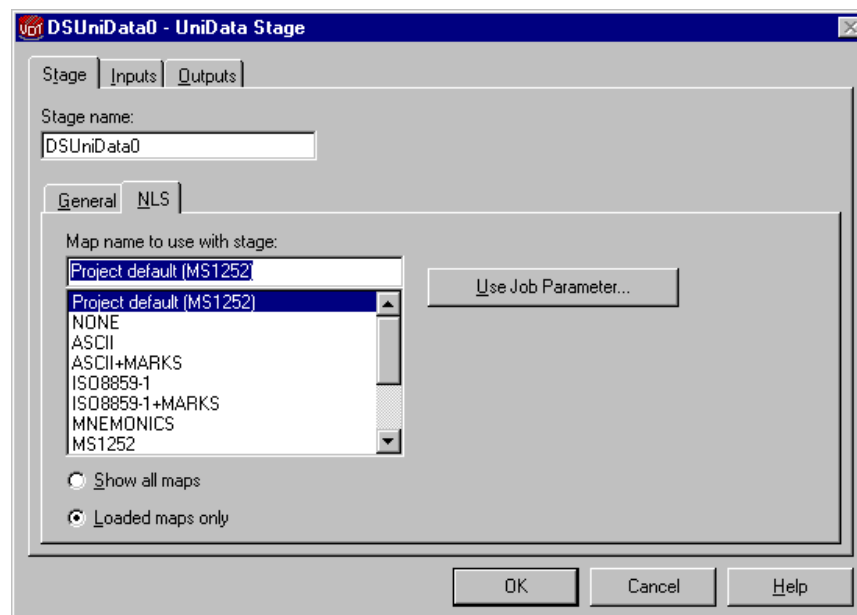
```
UniData Client error: call to UniOpenPos returned 45 - Client
version (11) and server version (12) are incompatible
```

you need to edit the *UNIAPL.INI* file in the Windows directory on your client machine to change the value of the **PROTOCOL** variable to match the server version. So, in the case of the example error, you would need to change the variable value from 11 to 12:

```
PROTOCOL = 12
```

## Defining Character Set Maps

You can define a character set map for a UniData stage using the **NLS** page of the UniData Stage dialog box.

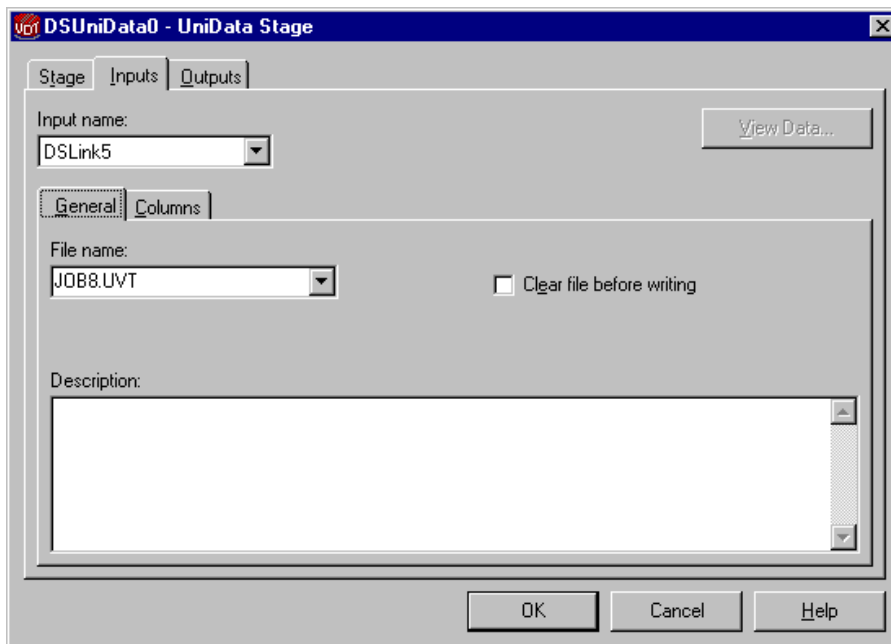


The default character set map (defined for the project or the job) can be changed by selecting a map name from the list. The page also has the following fields:

- **Show all maps.** Displays in the list all the maps supplied with DataStage. Maps cannot be used unless they have been loaded using the DataStage Administrator.
- **Loaded maps only.** Display the maps that are loaded and ready for use.
- **Use Job Parameter...** Allows you to specify a character set map as a parameter to the job containing the stage. If the parameter has not yet been defined, you are prompted to define it from the Job Properties dialog box (see page 5-30).

## Defining UniData Input Data

When you write data to a UniData file, the UniData stage has an input link. The properties of this link and the column definitions of the data are defined on the **Inputs** page in the UniData Stage dialog box.





The **Inputs** page has the following field and two pages:

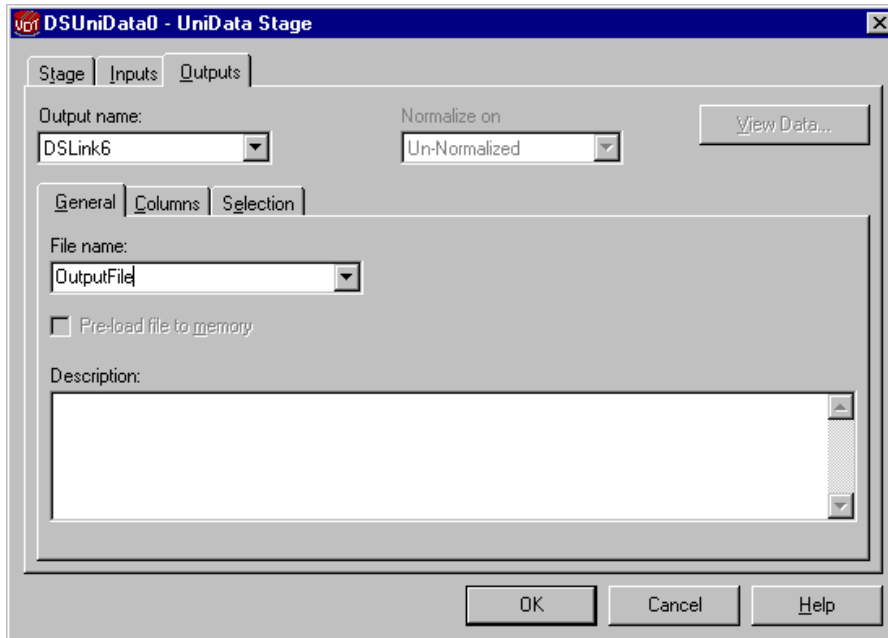
- **Input name.** The name of the input link. Choose the link you want to edit from the Input name drop-down list box. This list box displays all the input links to the UniData stage.
- **General.** Displayed by default. Contains the following fields and options:
  - **File name.** The name of the file the data is written to. You can use a job parameter to represent the file created during run time (see page 5-33 for more details), choose the file from the File name drop-down list box, or enter a file name. This list box contains all the files defined under the **Table Definitions > UniData > Server > Database** branch in the Manager.
  - Note:** If the file you want is not listed, you may need to define a table definition. For details on how to create a table definition, see Chapter 4, “Setting Up a Project.”
  - **Clear file before writing.** If you select this check box, the existing file is cleared and new data records are written to the empty file. This check box is cleared by default.
  - **Description.** Contains an optional description of the input link.
- **Columns.** Contains the column definitions for the data written to the file. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

**Note:** If you are using meta data that does not specify column positions, then the first column definition describes the UniData file’s key field. The remaining columns are ordered according to their position in the file. Do *not* reorder the column definitions in the grid unless you are certain you understand the consequences of your action.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the input link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

## Defining UniData Output Data

When you extract data from a UniData file, the UniData stage has an output link. The properties of this link and the column definitions of the data are defined on the **Outputs** page in the UniData Stage dialog box.



The **Outputs** page has the following two fields and three pages:

- **Output name.** The name of the output link. Choose the link you want to edit from the Output name drop-down list box. This list box displays all the output links from the UniData stage.
- **Normalize on.** This drop-down list box allows you to normalize (or unnest) data. You can normalize either on an association or on a single unassociated multivalued column. The Normalize on drop-down list box is only enabled for nonreference output links where meta data has been defined that contains multivalued fields.
- **General.** Displayed by default. Contains the following fields and options:
  - **File name.** The name of the file the data is read from. You can use a job parameter to represent the file created during run time (see “Specifying Job Parameters” on page 5-33 for more details) or choose the file from the File name drop-down list box. This list box contains all the files defined under the **Table Definitions > UniData > Server > Database** branch in the Repository.

**Note:** If the file you want is not listed, you need to define a table definition. For more information, see Chapter 4, “Setting Up a Project.”

- **Pre-load file to memory.** You can use this check box to improve performance if the output link is a reference input to a Transformer stage. If selected, the UniData file is read into memory when the job is run. This check box is cleared by default.
- **Description.** Contains an optional description of the output link.
- **Columns.** Contains the column definitions for the data on the chosen output link. For a description of how to enter or edit column definitions, see “Specifying Column Definitions” on page 5-18.
  - Note:** If you are using meta data that does not specify column positions, then the first column definition describes the UniData file’s key field. The remaining columns are ordered according to their position in the file. Do *not* reorder the column definitions in the grid unless you are certain you understand the consequences of your action.
- **Selection.** Contains optional SQL SELECT clauses for the conditional extraction of data from a file.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the output link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

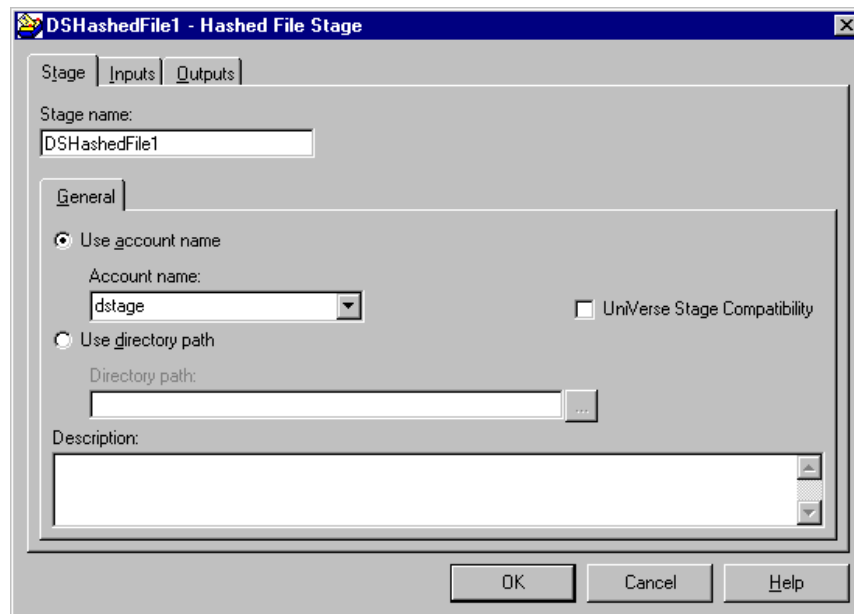


# 9

## Hashed File Stages

Hashed File stages represent a hashed file, i.e., a file that uses a hashing algorithm for distributing records in one or more groups on disk. You can use a Hashed File stage to extract or write data, or to act as an intermediate file in a job. The primary role of a Hashed File stage is as a reference table based on a single key field.

Each Hashed File stage can have any number of inputs or outputs. When you edit a Hashed File stage, the Hashed File Stage dialog box appears:



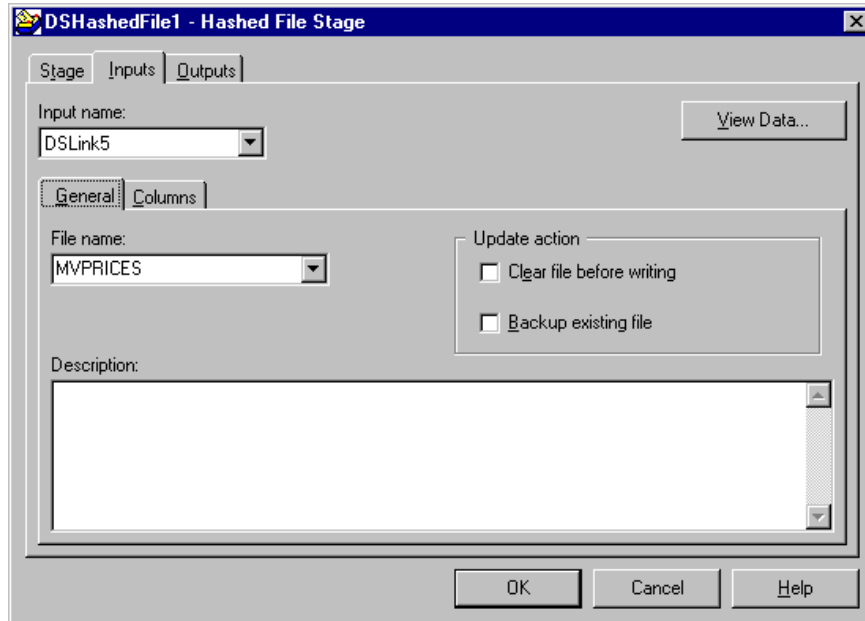
This dialog box can have up to three pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** Displays the name of the stage you are editing. This page has a **General** page, where you can enter text to describe the purpose of the stage in the **Description** field and specify where the data files are by clicking one of the option buttons:
  - **Use account name.** If you choose this option, you must choose the name of the account from the Account name drop-down list box. This list box contains all the accounts defined under the **Table Definitions > Hashed** branch in the DataStage Manager. If the account you want is not listed, you need to define a table definition. For details on how to create a table definition, see Chapter 4, “Setting Up a Project.” Alternatively, you can enter an account name or use a job parameter. For details on how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.
  - **Use directory path.** If you choose this option, you must specify a directory path containing the data file. The directory must be a UniVerse account and is used for UniVerse accounts that do not appear in the UV.ACCOUNT file. The directory is specified in the **Directory path** field. You can enter a path directly, click **Browse...** to search the system for a suitable directory, or use a job parameter. For more information about using **Browse...**, see “Browsing Server Directories” on page 5-23.
  - **UniVerse Stage Compatibility.** Select this check box to ensure that any job conversions will work correctly. With this option selected, the date or time will be represented in ISO format (depending on the Extended type) and numerics will be scaled according to the meta data. (The job conversion utility is a special standalone tool – it is not available within the DataStage Designer.)
- Note:** You cannot change the name of the stage from this dialog box. For details on how to change stage names, see “Renaming Stages” on page 5-14.
- **Inputs.** This page is only displayed if you have an input link to this stage. Specifies the data file to use and the associated column definitions for each data input link. This page also specifies how data is written to the data file.
- **Outputs.** This page is only displayed if you have an output link to this stage. Specifies the data file to use and the associated column definitions for each data output link.

Click **OK** to close this dialog box. Changes are saved when you save the job.

## Defining Hashed File Input Data

When you write data to a hashed file, the Hashed File stage has an input link. The properties of this link and the column definitions of the data are defined on the **Inputs** page in the Hashed File Stage dialog box.



The **Inputs** page has the following field and two pages:

- **Input name.** The name of the input link. Choose the link you want to edit from the Input name drop-down list box. This list box displays all the input links to the Hashed File stage.
- **General.** Displayed by default. Contains the following fields and options:
  - **File name.** The name of the file the data is written to. You can either use a job parameter to represent the file created during run time (see page 5-33 for more details) or choose the file from the File name drop-down list box. This list box contains all the files defined under the **Table Definitions > Hashed > Account name** branch in the Manager, where *Account name* is the name of the account chosen on the **Stage** page.

**Note:** If the file you want is not listed, you may need to define a table definition. For details on how to create a table definition, see Chapter 4, “Setting Up a Project.”

- **Clear file before writing.** If you select this check box, the existing file is cleared and new data records are written to the empty file. This check box is cleared by default.
- **Backup existing file.** If you select this check box, a backup copy of the existing file is made before the new data records are written to the file. The backup can be used to reset the file if a job is stopped or aborted at run time. See *DataStage Operator’s Guide* for more details. This check box is cleared by default.
- **Description.** Contains an optional description of the input link.
- **Columns.** Contains the column definitions for the data written to the file. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

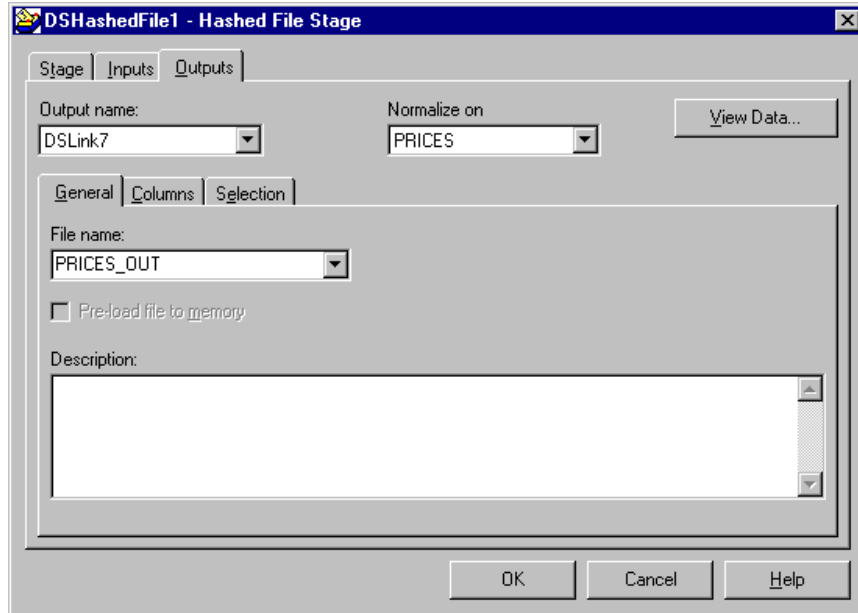
**Note:** If you are using meta data that does not specify column positions, then the first column definition describes the hashed file’s key field. The remaining columns are ordered according to their position in the file. Do *not* reorder the column definitions in the grid unless you are certain you understand the consequences of your action.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the input link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

## Defining Hashed File Output Data

When you extract data from a hashed file, the Hashed File stage has an output link. The properties of this link and the column definitions of the data are defined on the **Outputs** page in the Hashed File Stage dialog box.





The **Outputs** page has the following two fields and three pages:

- **Output name.** The name of the output link. Choose the link you want to edit from the Output name drop-down list box. This list box displays all the output links from the Hashed File stage.
- **Normalize on.** This drop-down list box allows you to normalize (or unnest) data. You can normalize either on an association or on a single unassociated multivalued column. The **Normalize on** drop-down list box is only enabled for nonreference output links where meta data has been defined that contains multivalued fields.
- **General.** Displayed by default. Contains the following fields and options:
  - **File name.** The name of the file the data is read from. You can use a job parameter to represent the file created during run time (see “Specifying Job Parameters” on page 5-33 for more details) or choose the file from the File name drop-down list box. This list box contains all the files defined under the **Table Definitions > Hashed > Account name** branch in the Repository, where *Account name* is the name of the account chosen on the **Stage** page.

**Note:** If the file you want is not listed, you need to define a table definition. For more information, see Chapter 4, “Setting Up a Project.”

- **Pre-load file to memory.** You can use this check box to improve performance if the output link is a reference input to a Transformer stage. If selected, the hashed file is read into memory when the job is run. This check box is cleared by default.
- **Description.** Contains an optional description of the output link.
- **Columns.** Contains the column definitions for the data on the chosen output link. For a description of how to enter or edit column definitions, see “Specifying Column Definitions” in page 5-18.

**Note:** If you are using meta data that does not specify column positions, then the first column definition describes the hashed file’s key field. The remaining columns are ordered according to their position in the file. Do *not* reorder the column definitions in the grid unless you are certain you understand the consequences of your action.

- **Selection.** Contains optional SQL SELECT clauses for the conditional extraction of data from a file. This page is only available if you have specified the hashed file by account name, rather than directory path, on the **Stage** page.

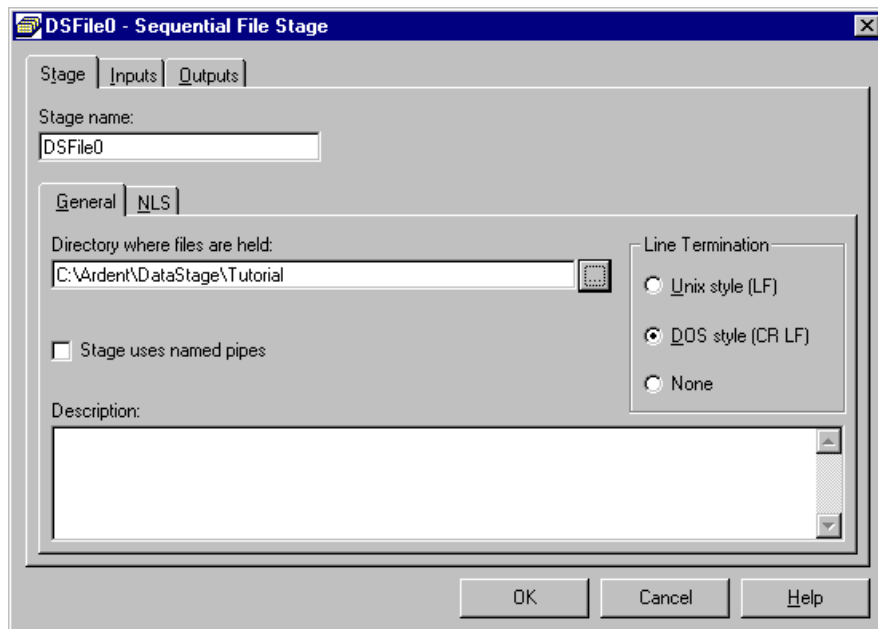
Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the output link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

# 10

## Sequential File Stages

Sequential File stages are used to extract data from, or write data to, a text file. The text file can be created or exist on any drive that is either local or mapped to the server. Each Sequential File stage can have any number of inputs or outputs.

When you edit a Sequential File stage, the Sequential File Stage dialog box appears:



This dialog box can have up to three pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** Displays the name of the stage you are editing. The **General** page allows you to enter the location of the files, line termination options, and an optional description of the stage.

You must choose the directory containing the files you want to read or write to. The operating system directory is specified in the **Directory where files are held** field. You can edit this field in two ways:

- Enter the directory path in the field. You can also include a job parameter in the directory path. For details of how to define and use job parameters, see “Specifying Job Parameters” on page 5-33.
- Click **Browse...** . The Browse directories dialog box appears. Expand the file system to display the directory you want. Highlight the directory and click **OK**. The **Directory where files are held** field is updated with the chosen directory. For more information about using **Browse...**, see “Browsing Server Directories” on page 5-23.

The line termination options let you set the type of line terminator to use in the Sequential File stage. By default, line termination matches the type used on your DataStage server. To change the value, choose one of **Unix style (LF)**, **DOS style (CR LF)**, or **None**.

Select the **Stage uses named pipes** check box if you want to make use of the named pipe facilities. These allow you to split up a large job into a number of smaller jobs. You may want to do this where there is a large degree of parallelism in your design, as it will increase performance and allow several developers to work on the design at the same time. With this check box selected, all inputs and outputs to the stage use named pipes, and you cannot specify a directory in the **Directory where files are held** field. Examples of how to use the named pipe facilities are given in the online Help.

**Note:** You cannot change the name of the stage from this dialog box. For details on changing stage names, see “Renaming Stages” on page 5-14.

If NLS is enabled, the **NLS** page allows you to define character set mapping and UNICODE settings for the stage. For more information, see “Defining Character Set Maps” on page 10-3.

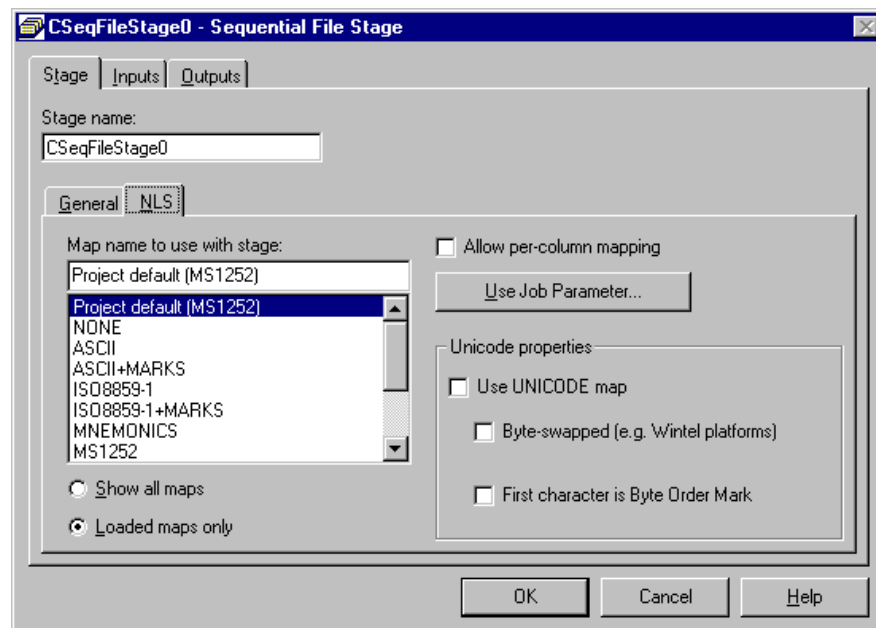
- **Inputs.** Contains information about the file formats and column definitions for each data input link. This page is displayed only if you have an input link to this stage.

- **Outputs.** Contains information about the file format and column definitions for the data output links. This page is displayed only if you have an output link to this stage.

Click **OK** to close this dialog box. Changes are saved when you save the job.

## Defining Character Set Maps

You can define a character set map for a Sequential File stage using the **NLS** page of the Sequential File Stage dialog box.



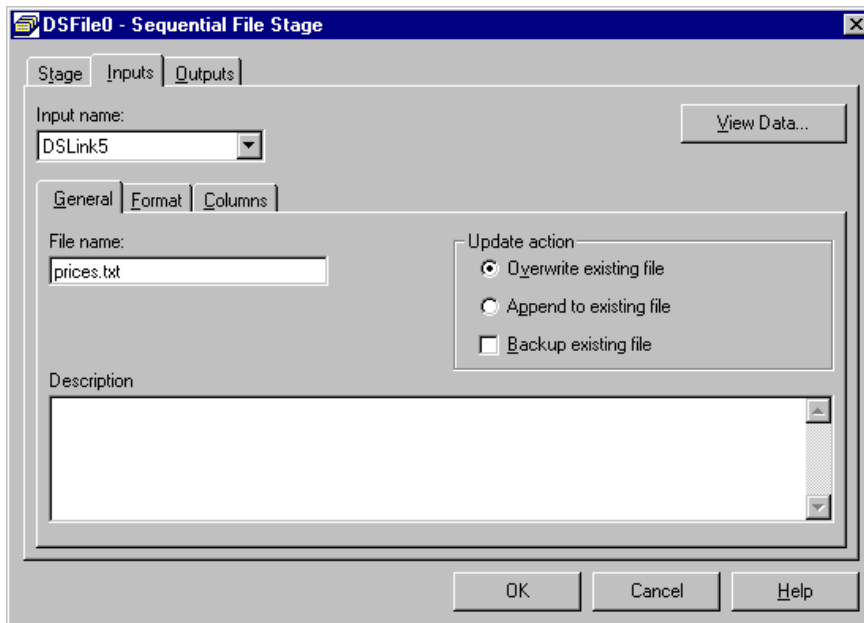
The default character set map (defined for the project or the job) can be changed by selecting a map name from the list. The page also has the following fields:

- **Show all maps.** Choose this to display all the maps supplied with DataStage in the list. Maps cannot be used unless they have been loaded using the DataStage Administrator.
- **Loaded maps only.** Displays the maps that are loaded and ready for use.

- **Use Job Parameter...** . Allows you to specify a character set map as a parameter to the job containing the stage. If the parameter has not yet been defined, you are prompted to define it from the Job Properties dialog box (see page 5-30).
- **Use UNICODE map.** If you select this, the character set map is overridden, and all data is read and written in UNICODE format with two bytes per character.
  - If **Byte swapped** is selected, the data is read or written with the lower-order byte first. For example, 0X0041 (that is, “A”) is written as bytes 0X41,0X00. Otherwise it is written as 0X00,0X41.
  - If **First character is Byte Order Mark** is selected, the stage reads or writes the sequence 0XFE,0XFF if byte swapped, or 0XFF,0XFE if not byte swapped.

## Defining Sequential File Input Data

When you write data to a sequential file, the Sequential File stage has an input link. The properties of this link and the column definitions of the data are defined on the **Inputs** page in the Sequential File Stage dialog box.



The **Inputs** page has the following field and three pages:

- **Input name.** The name of the input link. Choose the link you want to edit from the **Input name** drop-down list box. This list box displays all the input links to the Sequential File stage.
- **General.** Displayed by default. Contains the following parameters:
  - **File name.** The name of the file the data is written to. You can enter a job parameter to represent the file created during run time. For details on how to define job parameters, see “Specifying Job Parameters” on page 5-33.
  - **Description.** Contains an optional description of the input link.

The **General** page also contains options that determine how the data is written to the file. These are displayed under the Update action area:

- **Overwrite existing file.** This is the default option. If this option button is selected, the existing file is truncated and new data records are written to the empty file.
- **Append to existing file.** If you select this option button, the data records are appended to the end of the existing file.
- **Backup existing file.** If you select this check box, a backup copy of the existing file is taken. The new data records are written based on whether you chose to append to or overwrite the existing file.

**Note:** The backup can be used to reset the file if a job is stopped or aborted at run time. See *DataStage Operator's Guide* for more details.

- **Format.** Contains parameters that determine the format of the data in the file. There are up to five check boxes:
  - **Fixed-width columns.** If you select this check box, the data is written to the file in fixed-width columns. The width of each column is specified by the SQL display size (set in the **Display** column in the Columns grid). This option is cleared by default.
  - **First line is column names.** Select this check box if the first row of data in the file contains column names. This option is cleared by default, that is, the first row in the file contains data.
  - **Omit last new-line.** Select this check box if you want to remove the last newline character in the file. This option is cleared by default, i.e., the newline character is not removed.

- **Check data against meta data.** This is only enabled if you have selected **Fixed-width columns**. Select this to check data in the file against the meta data. A message is written to the log if a final column in a row is found to be longer or shorter than the length specified by the **Length** field for that column.
- **Flush after every row.** This only appears if you have selected **Stage uses named pipes** on the **Stage** page. Selecting this check box causes data to be passed between the reader and writer of the pipe one record at a time.

There are up to six fields on the **Format** page:

- **Delimiter.** Contains the delimiter that separates the data fields in the file. By default this field contains a comma. You can enter a single printable character (other than a digit) or a number from 0 through 255 to represent the ASCII code for the character you want to use.
- **Quote character.** Contains the character used to enclose a data value that contains the delimiter character as data. By default this field contains a double quotation mark. You can enter a single printable character (other than a digit) or a number from 1 through 255 to represent the ASCII code for the character you want to use. You can suppress the quote character by entering 0.
- **Spaces between columns.** This field is only active when you select the **Fixed-width columns** check box. Contains a number to represent the number of spaces used between columns.
- **NULL string.** Contains characters that are written to the file when a column contains an SQL NULL.

The following fields appear only if you have selected **Stage uses named pipes** on the **Stage** page:

- **Wait for reader timeout.** Specifies how long the stage will wait for a connection when reading from a pipe before timing out. Recommended values are between 30 and 600 seconds. If the stage times out an error is raised and the job is aborted.
- **Write timeout.** Specifies how long the stage will attempt to write data to a pipe before timing out. Recommended values are between 30 and 600 seconds. If the stage times out an error is raised and the job is aborted.
- **Columns.** Contains the column definitions for the data on the chosen input link. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

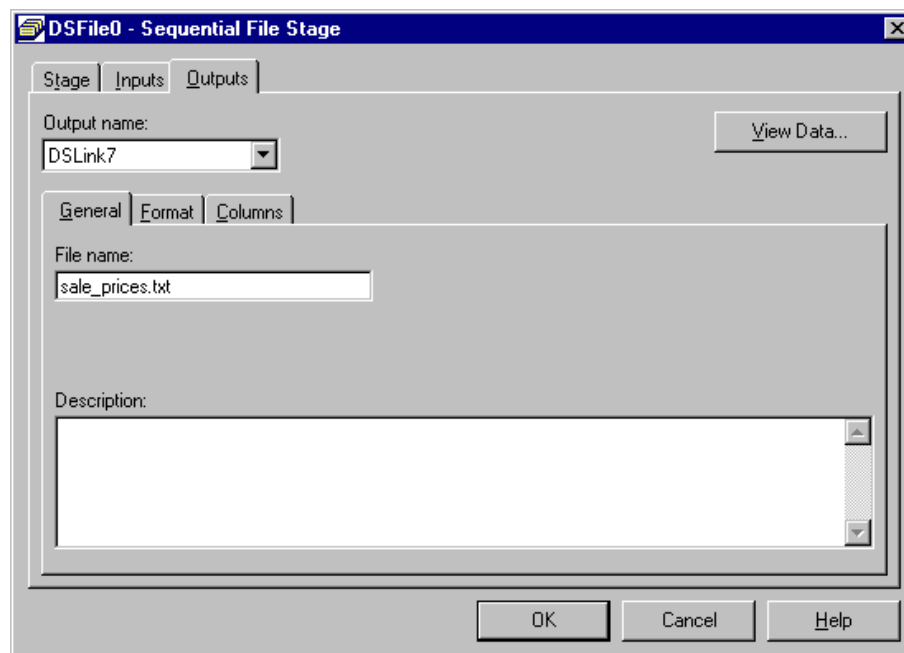


**Note:** The SQL data type properties affect how data is written to a sequential file. The SQL display size determines the size of fixed-width columns. The SQL data type determines how the data is justified in a column: character data types are quoted and left justified, numeric data types are not quoted and are right justified. The SQL properties are in the Columns grid when you edit an input column.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the input link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.

## Defining Sequential File Output Data

When you extract (read) data from a sequential file, the Sequential File stage has an output link. The properties of this link and the column definitions of the data are defined on the **Outputs** page in the Sequential File Stage dialog box.



The **Outputs** page has the following field and three pages:

- **Output name.** The name of the output link. Choose the link you want to edit from the Output name drop-down list box. This list box displays all the output links to the Sequential File stage.
- **General.** Displayed by default. There are two fields:
  - **File name.** The name of the file the data is extracted from. You can enter a job parameter to represent the file created during run time. For details on how to define job parameters, see “Specifying Job Parameters” on page 5-33.
  - **Description.** Contains an optional description of the output link.
- **Format.** Contains parameters that determine the format of the data in the file. There are up to three check boxes:
  - **Fixed-width columns.** If you select this check box, the data is extracted from the file in fixed-width columns. The width of each column is specified by the SQL display size (set in the **Display** column in the Columns grid). This option is cleared by default.
  - **First line is column names.** Select this check box if the first row of data in the file contains column names. This option is cleared by default, that is, the first row in the file contains data.
  - **Check data against meta data.** This is only enabled if you have selected **Fixed-width columns**. Select this to check data in the file against the meta data. A message is written to the log if a final column in a row is found to be longer or shorter than the length specified by the **Length** field for that column.

There are up to six fields on the **Format** page:

- **Delimiter.** Contains the delimiter that separates the data fields in the file. By default this field contains a comma. You can enter a single printable character (other than a digit) or a number 0 through 255 to represent the ASCII code for the character you want to use.
- **Quote character.** Contains the character used to enclose a data value that contains the delimiter character as data. By default this field contains a double quotation mark. You can enter a single printable character (other than a digit) or a number 0 through 255 to represent the ASCII code for the character you want to use.

- **Spaces between columns.** This field is only active when you select the **Fixed-width columns** check box. Contains a number to represent the number of spaces used between columns.
- **NULL string.** Contains characters which, when encountered in a sequential file being read, are interpreted as the SQL NULL value.

The following fields appear only if you have selected **Stage uses named pipes** on the **Stage** page:

- **Wait for writer timeout.** Specifies how long the stage will wait for a connection when writing to a pipe before timing out. Recommended values are between 30 and 600 seconds. If the stage times out an error is raised and the job is aborted.
- **Read timeout.** Specifies how long the stage will attempt to read data from a pipe before timing out. Recommended values are between 30 and 600 seconds. If the stage times out an error is raised and the job is aborted.
- **Columns.** Contains the column definitions for the data on the chosen output link. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

Click the **View Data...** button to invoke the Data Browser. This enables you to look at the data associated with the output link. For a description of the Data Browser, see “Using the Data Browser” on page 5-24.



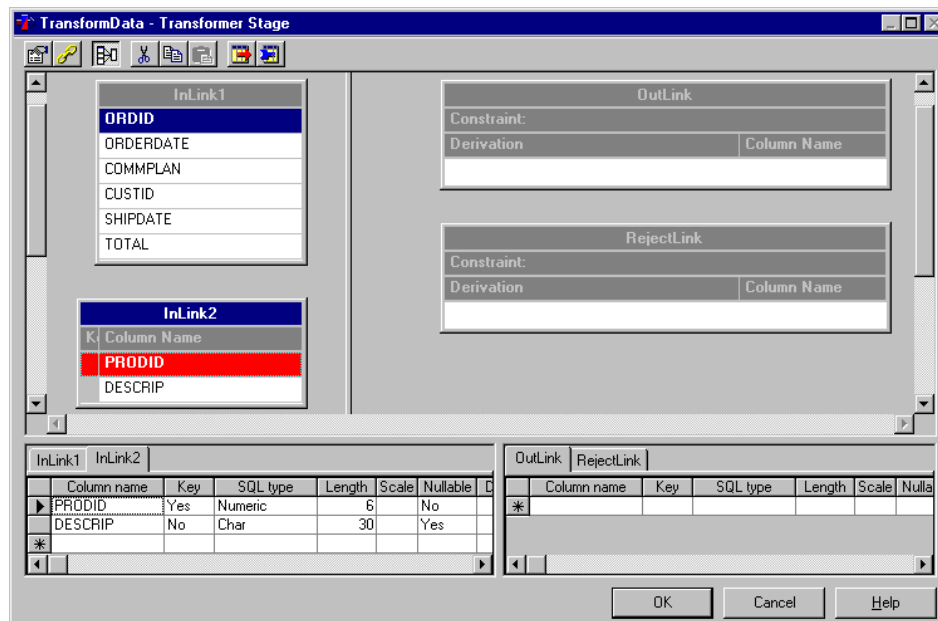
# 11

## Transformer Stages

Transformer stages do not extract data or write data to a target database. They are used to handle extracted data, perform any conversions required, and pass data to another Transformer stage or a stage that writes data to a target data table.

Transformer stages can have any number of inputs and outputs. The link from the main data input source is designated the primary input link. There can only be one primary input link, but there can be any number of reference inputs.

When you edit a Transformer stage, the Transformer Editor appears. For an example Transformer stage with two inputs and two outputs, the Transformer Editor appears as follows. In this example, meta data has previously been defined for the input links, but not the output links.

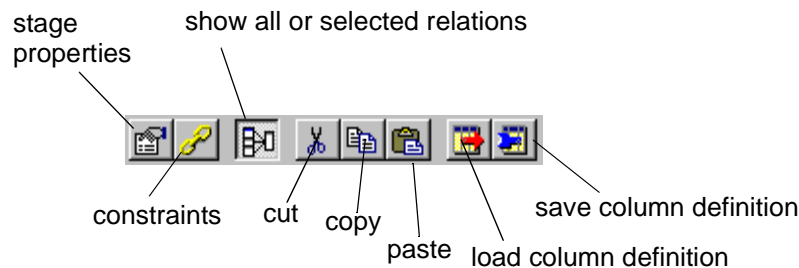


## Transformer Editor Components

The Transformer Editor has the following components.

### Toolbar

The Transformer toolbar contains the following icons:



### Link Area

The top area displays links to and from the Transformer stage, showing their columns and the relationships between them.

The link area is where all column definitions and key expressions are defined.

The link area is divided into two panes; you can drag the splitter bar between them to resize the panes relative to one another. There is also a horizontal scroll bar, allowing you to scroll the view left or right.

The left pane shows input links, the right pane shows output links. The input link shown at the top of the left pane is always the primary link. Any subsequent links are reference links. For all types of link, key fields are shown in bold. Reference link key fields that have no expression defined are shown in red, as are output columns that have no derivation defined.

Within the Transformer Editor, a single link may be selected at any one time. When selected, the link's title bar is highlighted.

### Meta Data Area

The bottom area shows the column meta data for input and output links. Again this area is divided into two panes: the left showing input link meta data and the right showing output link meta data.

The meta data for each link is shown in a grid contained within a tabbed page. Click the tab to bring the required link to the front. That link is also selected in the link area.

If you select a link in the link area, its meta data page is brought to the front automatically.

You can edit the grids to change the column meta data on any of the links. You can also add and delete meta data.

## Shortcut Menus

The Transformer Editor shortcut menus are invoked by right-clicking the links in the links area.

There are two slightly different menus, depending on whether you right-click an input link or an output link. The input link menu offers you operations on key expressions; the output link menu offers you operations on derivations.

The shortcut menu enables you to:

- Edit, validate, or clear a key expression or derivation.
- Append a new column to the selected link.
- Find a particular column.
- Select all columns on a link.
- Insert or delete columns.
- Cut, copy, and paste a column or a key expression or a derivation.

If you invoke the menu from a link header, you can:

- Invoke the Properties dialog box to enter a description of the link.
- Invoke the Constraints dialog box to specify a constraint (only available for output links).

Right-clicking in the links area background invokes the background shortcut menu. This menu allows you to invoke the Stage Properties dialog box in order to specify a before- or after-stage subroutine, or the Constraints dialog box in order to specify a constraint for the selected output link. This menu also allows you to toggle between viewing link relations for all links, or for the selected link only.

Right-clicking in the meta data area of the Transformer Editor invokes the standard grid editing shortcut menus.

## Transformer Stage Basic Concepts

When you first edit a Transformer stage, it is likely that you will have already defined what data is input to the stage on the input links. You will use the Transformer Editor to define the data that will be output by the stage and how it will be transformed. (You can define input data using the Transformer Editor if required.)

This section explains some of the basic concepts of using a Transformer stage.

### Input Links

The main data source is joined to the Transformer stage via the primary link, but the stage can also have any number of reference input links.

A reference link represents a table lookup. These are used to provide information that might affect the way the data is changed, but do not supply the actual data to be changed.

Reference input columns can be designated as key fields. You can specify key expressions that are used to evaluate the key fields. The most common use for the key expression is to specify an equijoin, which is a link between a primary link column and a reference link column. For example, if your primary input data contains names and addresses, and a reference input contains names and phone numbers, the reference link **name** column is marked as a key field and the key expression refers to the primary link's **name** column. During processing, the name in the primary input is looked up in the reference input. If the names match, the reference data is consolidated with the primary data. If the names do not match, i.e., there is no record in the reference input whose key matches the expression given, all the columns specified for the reference input are set to the null value.

### Output Links

You can have any number of output links from your Transformer stage.

You may want to pass some data straight through the Transformer stage unaltered, but it's likely that you'll want to transform data from some input columns before outputting it from the Transformer stage.

You can specify such an operation by entering a BASIC expression or by selecting a transform function to apply to the data. DataStage has many built-in transforms, or you can define your own custom transforms that are stored in the Repository and can be reused as required.



The source of an output link column is defined in that column's **Derivation** cell within the Transformer Editor. You can use the Expression Editor to enter expressions or transforms in this cell. You can also simply drag an input column to an output column's **Derivation** cell, to pass the data straight through the Transformer stage.

In addition to specifying derivation details for individual output columns, you can also specify constraints that operate on entire output links. A constraint is a BASIC expression that specifies criteria that data must meet before it can be passed to the output link. You can also specify a reject link, which is an output link that carries all the data not output on other links, that is, columns that have not met the criteria.

Each output link is processed in turn. If the constraint expression evaluates to TRUE for an input row, the data row is output on that link. Conversely, if a constraint expression evaluates to FALSE for an input row, the data row is not output on that link.

Constraint expressions on different links are independent. If you have more than one output link, an input row may result in a data row being output from some, none, or all of the output links.

For example, if you consider the data that comes from a paint shop, it could include information about any number of different colors. If you want to separate the colors into different files, you would set up different constraints. You could output the information about green and blue paint on LinkA, red and yellow paint on LinkB, and black paint on LinkC.

When an input row contains information about yellow paint, the LinkA constraint expression evaluates to FALSE and the row is not output on LinkA. However, the input data does satisfy the constraint criterion for LinkB and the rows are output on LinkB.

If the input data contains information about white paint, this does not satisfy any constraint and the data row is not output on Links A, B or C, but will be output on the reject link. The reject link is used to route data to a table or file that is a "catch-all" for rows that are not output on any other link. The table or file containing these rejects is represented by another stage in the job design.

## **Before-Stage and After-Stage Routines**

Because the Transformer stage is an active stage type, you can specify routines to be executed before or after the stage has processed the data. For example, you might use a before-stage routine to prepare the data before processing starts. You might use an after-stage routine to send an electronic message when the stage has finished.

## Editing Transformer Stages

The Transformer Editor enables you to perform the following operations on a Transformer stage:

- Create new columns on a link
- Delete columns from within a link
- Move columns within a link
- Edit column meta data
- Define output column derivations
- Define input column key expressions
- Specify before- and after-stage subroutines
- Define link constraints and handle rejects

## Using Drag and Drop

Many of the Transformer stage edits can be made simpler by using the Transformer Editor's drag and drop functionality. You can drag columns from any link to any other link. Common uses are:

- Copying input columns to output links
- Moving columns within a link
- Copying derivations in output links
- Copying key expressions in input links

To use drag and drop:

1. Click the source cell to select it.
2. Click the selected cell again and, without releasing the mouse button, drag the mouse pointer to the desired location within the target link. An insert point appears on the target link to indicate where the new cell will go.
3. Release the mouse button to drop the selected cell.

You can drag and drop multiple columns, key expressions, or derivations. Use the standard Explorer keys when selecting the source column cells, then proceed as for a single cell.

You can add a column to the end of an existing derivation or key expression by holding down the **Ctrl** key as you drag the column.

The drag and drop insert point is shown below:

Outlink	
Constraint:	
Derivation	Column Name
Inlink1.PRODID	PRODID
Inlink1.STDPRICE	STDPRICE
Inlink1.MINPRICE	MINPRICE
Inlink1.STARTDATE	STARTDATE
Inlink1.ENDDATE	ENDDATE

## Specifying the Primary Input Link

The first link to a Transformer stage is always designated as the primary input link. However, you can choose an alternative link to be the primary link if necessary. To do this:

1. Select the current primary input link in the Diagram window.
2. Choose **Convert to Reference** from the Diagram window shortcut menu.
3. Select the reference link that you want to be the new primary input link.
4. Choose **Convert to Stream** from the Diagram window shortcut menu.

## Creating and Deleting Columns

You can create columns on links to the Transformer stage using any of the following methods:

- Select the link, then click the **load column definition** icon in the toolbar to invoke the standard load columns dialog box.
- Use drag and drop or copy and paste functionality to create a new column by copying from an existing column on another link.
- Use the shortcut menus to create a new column definition.
- Edit the grids in the link's meta data page to insert a new column.

When copying columns, a new column is created with the same meta data as the column it was copied from.

To delete a column from within the Transformer Editor, select the column you want to delete and click the **cut** icon or choose **Delete Column** from the shortcut menu.

## Moving Columns Within a Link

You can move columns within a link using either drag and drop or cut and paste. Select the required column, then drag it to its new location, or cut it and paste it in its new location.

## Editing Column Meta Data

You can edit column meta data from within the grid in the bottom of the Transformer Editor. Select the page for the link meta data that you want to edit, then use the standard DataStage edit grid controls.

The meta data shown does not include column derivations or key expressions, since these are edited in the links area.

## Defining Output Column Derivations

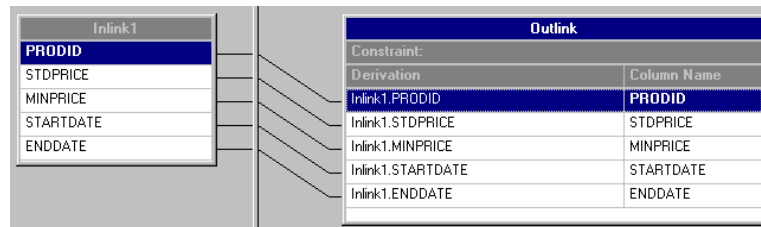
You can define the derivation of output columns from within the Transformer Editor in three ways:

- If you require an output column to be directly derived from an input column, with no transformations performed, then you can use drag and drop or copy and paste to copy an input column to an output link. The output columns will have the same names as the input columns from which they were derived.
- You may need one output link column derivation to be the same as another output link column derivation. In this case you can use drag and drop or copy and paste to copy the derivation cell from one column to another.
- In many cases you will need to transform data before deriving an output column from it. For these purposes you can use the Expression Editor. To display the Expression Editor, double-click on the required output link column **Derivation** cell. (You can also invoke the Expression Editor using the shortcut menu or the shortcut keys.)

If a derivation is displayed in red it means that the Transformer Editor considers it incorrect. (In some cases this may simply mean that the derivation does not meet the strict usage pattern rules of the DataStage engine, but will actually function correctly.)

Once an output link column has a derivation defined that contains any input link columns, then a relationship line is drawn between the input column and the output column, as shown in the following example. This is a simple example; there can be multiple relationship lines either in or out of columns. You can choose

whether to view the relationships for all links, or just the relationships for the selected links, using the icon in the toolbar.



## Defining Input Column Key Expressions

You can define key expressions for key fields of reference inputs. This is similar to defining derivations for output columns.

In most cases a key expression will be an equijoin from a primary input link column. You can specify an equijoin in two ways:

- Use drag and drop to drag a primary input link column to the appropriate key expression cell.
- Use copy and paste to copy a primary input link column and paste it on the appropriate key expression cell.

A relationship link is drawn between the primary input link column and the key expression.

You can also use drag and drop or copy and paste to copy an existing key expression to another input column, and you can drag or copy multiple selections.

If you require a more complex expression than an equijoin, then you can double-click the required key expression cell to invoke the Expression Editor.

If a key expression is displayed in red it means that the Transformer Editor considers it incorrect. (In some cases this may simply mean that the key expression does not meet the strict usage pattern rules of the DataStage engine, but will actually function correctly.)

Initially, key expression cells occupy a very narrow column. In most cases the relationship line gives sufficient information about the key expression, but otherwise you can drag the left edge of the column to expand it.

Inlink1	
	PRODID
	STDPRI
	MINPRICE
	STARTDATE
	ENDDATE

Inlink2	
Key Expression	Column Name
Inlink1.PRODID	PRODID
	DESCRIP

## Specifying Before-Stage and After-Stage Subroutines

Because the Transformer stage is an active stage type, you can specify routines to be executed before or after the stage has processed the data.

To specify a routine, click the stage properties icon in the toolbar to invoke the following dialog box:

The dialog box titled "TransformData - Transformer Stage Properties" contains the following fields:

- Stage name:** A text box containing "TransformData".
- Description:** A large empty text area.
- Before-stage subroutine:** A dropdown menu with "(none)" selected.
- Input Value:** A text box next to the before-stage subroutine dropdown.
- After-stage subroutine:** A dropdown menu with "(none)" selected.
- Input Value:** A text box next to the after-stage subroutine dropdown.
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom.

The dialog box contains the following fields:

- **Before-stage subroutine** and **Input Value**. Contain the name (and value) of a subroutine that is executed before the stage starts to process any data.
- **After-stage subroutine** and **Input Value**. Contain the name (and value) of a subroutine that is executed after the stage has processed the data.

Choose a routine from the drop-down list box. This list box contains all the built routines defined as a **Before/After Subroutine** under the **Routines** branch in the Repository. Enter an appropriate value for the routine's input argument in the **Input Value** field.

If you choose a routine that is defined in the Repository, but which was edited but not compiled, a warning message reminds you to compile the routine when you close the Transformer stage dialog box.

If you installed or imported a job, the **Before-stage subroutine** or **After-stage subroutine** field may reference a routine that does not exist on your system. In this case, a warning message appears when you close the dialog box. You must install or import the "missing" routine or choose an alternative one to use.

If you edit a job created using Release 1 of DataStage, the **Before-stage subroutine** or **After-stage subroutine** field may contain the name of a routine created at Release 1. When DataStage is upgraded, these routines are identified and automatically renamed. For example, if you used a before-stage subroutine called BeforeSubr, this appears as BeforeSubr\<Rev1> in the **Before-stage subroutine** field. You can continue to use these routines. However, because you could not specify input values for routines at Release 1 of DataStage, the **Input Value** field grays out when you use one of these "old" routines.

## Defining Constraints and Handling Rejects

You can define limits for output data by specifying a constraint. Constraints are BASIC expressions and you can specify a constraint for each output link from a Transformer stage. You can also specify that a particular link is to act as a reject link. Reject links output rows that have not been written on any other output links from the transformer stage.

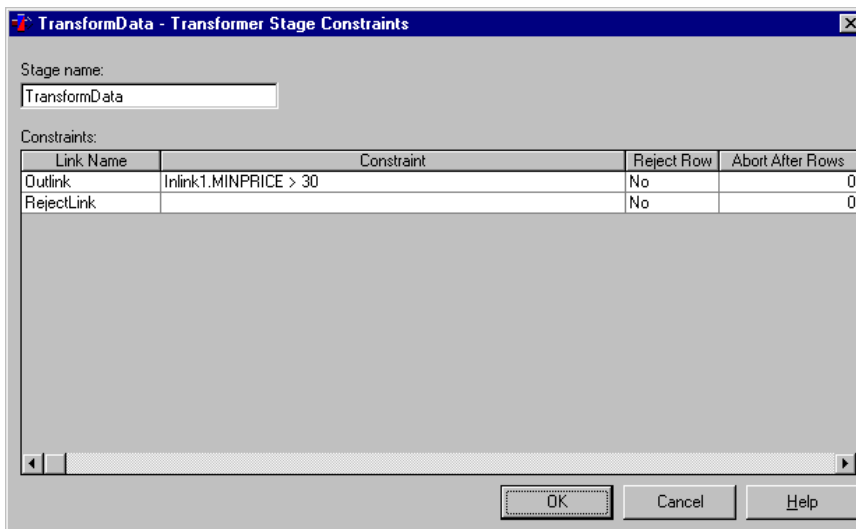
To define a constraint or specify a reject link, do one of the following:

- Select an output link and click the constraints icon.
- Double-click the output link's constraint entry field.
- Choose **Constraints** from the background or header shortcut menus.

A dialog box appears which allows you either to define constraints for any of the Transformer output links or to define a link as a reject link.

Define a constraint by entering a BASIC expression in the **Constraint** field for that link. Once you have done this, any constraints will appear below the link's title bar in the Transformer Editor.

Define a reject link by leaving the **Constraint** field blank, and choosing **Yes** in the **Reject Row** field. If you have defined a link as a reject link, then **[Reject]** appears below the link's title bar.



The **Constraint** field is left blank when you specify a reject link. You can, however, select **Yes** in the **Reject Row** field of a link for which you have specified a constraint. This causes the number of rows written to that link (i.e., rows that satisfy the constraint) to be recorded in the job log file at run time.

## The DataStage Expression Editor

The DataStage Expression Editor helps you to enter correct expressions when you edit Transformer stages. It also helps you to define custom transforms within the DataStage Manager. The Expression Editor can:

- Facilitate the entry of expression elements
- Complete the names of frequently used variables
- Validate variable names and the complete expression



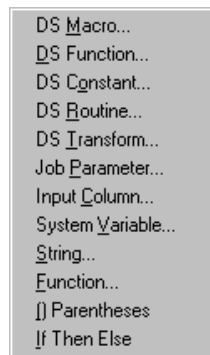
The Expression Editor can be invoked from:

- Output link **Derivation** cells
- Input link **Key Expression** cells
- Constraint dialog box
- Transform dialog box in the DataStage Manager

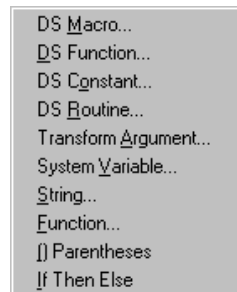
## Entering Expressions

Whenever the insertion point is in an expression box, you can use the Expression Editor to suggest the next element in your expression. Do this by right-clicking the box, or by clicking the **Suggest** button to the right of the box. This invokes the Suggest Operand or Suggest Operator menu. Which menu appears depends on context, i.e., whether you should be entering an operand or an operator as the next expression element.

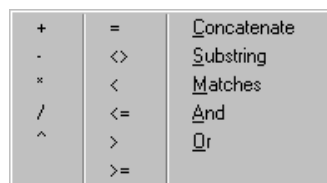
You will be offered a different selection on the Suggest Operand menu depending on whether you are defining key expressions, derivations and constraints, or a custom transform. The Suggest Operator menu is always the same.



**Suggest Operand Menu**  
Transformer Stage



**Suggest Operand Menu**  
Defining Custom Transforms



**Suggest Operator Menu**

## Completing Variable Names

The Expression Editor stores variable names. When you enter a variable name you have used before, you can type the first few characters, then press **F5**. The Expression Editor completes the variable name for you.

If you enter the name of an input link followed by a period, for example, **DailySales.**, the Expression Editor displays a list of the column names of that link. If you continue typing, the list selection changes to match what you type. You can also select a column name using the mouse. Enter a selected column name into the expression by pressing **Tab** or **Enter**. Press **Esc** to dismiss the list without selecting a column name.

## Validating the Expression

When you have entered an expression in the Transformer Editor, press **Enter** to validate it. The Expression Editor checks that the syntax is correct and that any variable names used are acceptable to the compiler. When using the Expression Editor to define a custom transform, click **OK** to validate the expression.

If there is an error, a message appears and the element causing the error is highlighted in the expression box. You can either correct the expression or close the Transformer Editor or Transform dialog box.

Within the Transformer Editor, the invalid expressions are shown in red. (In some cases this may simply mean that the expression does not meet the strict usage pattern rules of the DataStage engine, but will actually function correctly.)

For more information about the syntax you can use in an expression, see DataStage Developer's Help.

## Exiting the Expression Editor

You can exit the Expression Editor in the following ways:

- Press **Esc** (which discards changes).
- Press **Return** (which accepts changes).
- Click outside the Expression Editor box (which accepts changes).

## Configuring the Expression Editor

The Expression Editor is switched on by default. If you prefer not to use it, you can switch it off or use selected features only. The Expression Editor is configured by editing the Designer options. For more information, see "Specifying Designer Options" on page 5-45.

## Transforms

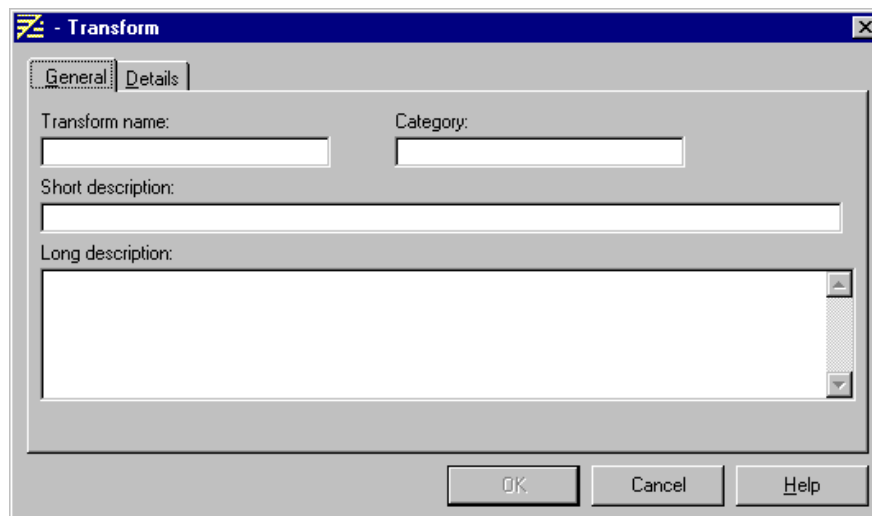
Transforms are used in the Transformer stage to convert your data to a format you want to use in the final data mart. Each transform specifies the BASIC function used to convert the data from one type to another. There are a number of built-in transforms supplied with DataStage, which are described in Appendix A, “Built-In Transforms and Routines.”

If the built-in transforms are not suitable or you want a specific transform to act on a specific data element, you can create custom transforms in the DataStage Manager.

To create a custom transform:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Select the **Transforms** branch in the project tree and do one of the following:
  - Choose **File ► New Transform...**
  - Choose **New Transform...** from the shortcut menu.
  - Click the **New** icon on the toolbar.

The Transform dialog box appears:



The screenshot shows a dialog box titled "Transform" with a blue title bar. It has two tabs: "General" (selected) and "Details". The "General" tab contains the following fields:

- "Transform name:" with a text input field.
- "Category:" with a text input field.
- "Short description:" with a single-line text input field.
- "Long description:" with a multi-line text area.

At the bottom of the dialog box are three buttons: "OK", "Cancel", and "Help".

This dialog box has two pages:

- **General.** Displayed by default. Contains general information about the transform.
  - **Details.** Allows you to specify source and target data elements, the function, and arguments to use.
3. Enter the name of the transform in the **Transform name** field. This name is used to create a leaf under the category branch. The name entered here must be unique; as no two transforms can have the same name. Also note that the transform should not have the same name as an existing BASIC function; if it does, the function will be called instead of the transform when you run the job.
  4. Enter a category name in the **Category** field. This name is used to create a branch under the main **Transforms** branch. If you do not enter a name in this field, the transform is created under the main **Transforms** branch.

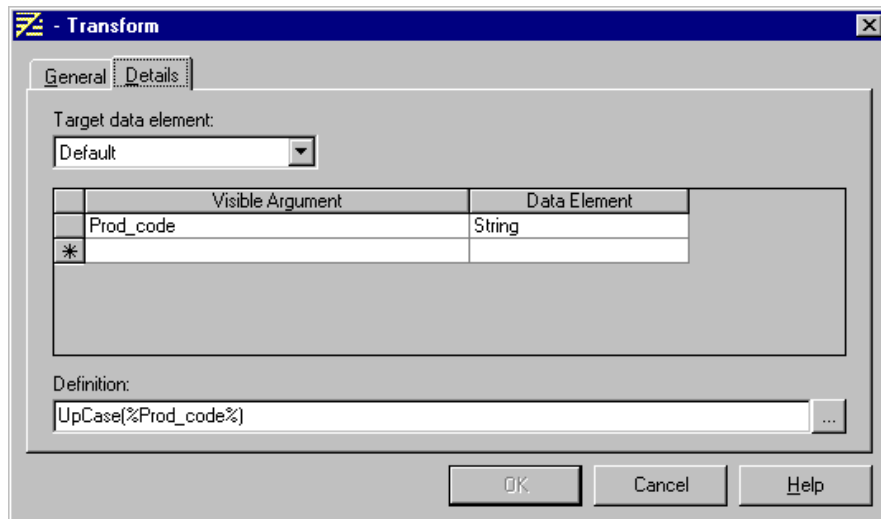
You can create more than one branch level for the category by including \ in the name. For example, if you enter `Custom\User`, the following branches are created:

**Transforms ► Custom ► User**

In this example, the new transform is created as a leaf under the **User** branch.

5. Enter a brief description of the transform in the **Short description** field. This is an optional field. The text entered here is displayed when you choose **View ► Details** from the DataStage Manager window.
6. Enter a detailed description of the transform in the **Long description** field. This is also an optional field. Once this page is complete, you can specify how the data is converted.

- Click the **Details** tab. The **Details** page appears at the front of the Transform dialog box:



- Choose the data element you want as the target data element from the **Target data element** drop-down list box.
- Specify the source arguments for the transform in the **Source Arguments** grid. Enter the name of the argument and choose the corresponding data element from the drop-down list.
- Use the Expression Editor in the **Definition** field to enter an expression which defines how the transform behaves.
- Click **OK** to save the transform and close the Transform dialog box. The new transform appears in the project tree under the specified branch.

You can then use the new transform from within the Transformer Editor.

**Note:** If NLS is enabled, avoid using the built-in ICONV and OCONV functions to map data unless you fully understand the consequences of your actions.



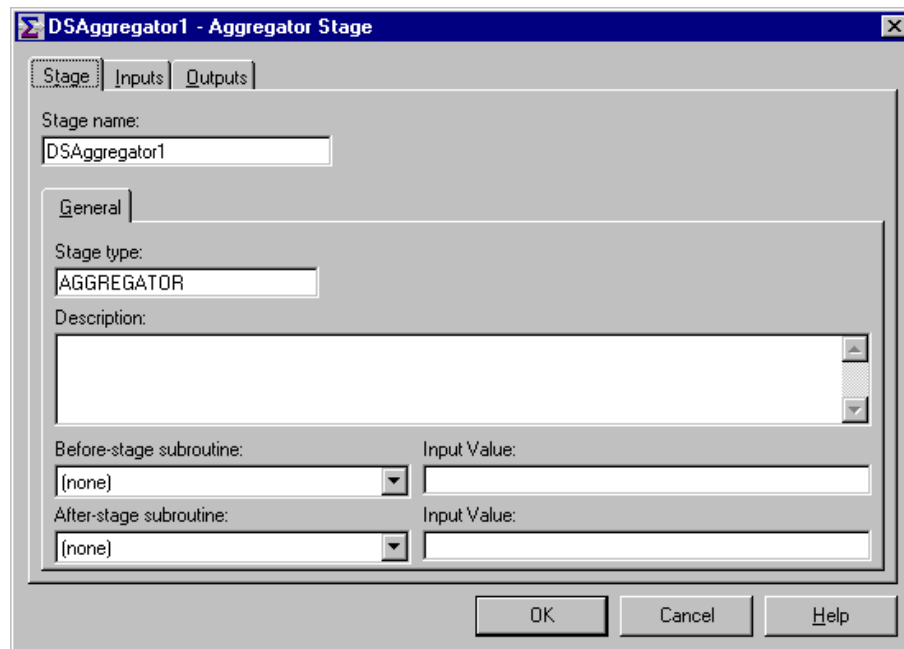
# 12

## Aggregator Stages

Aggregator stages classify data rows from a single input link into groups and compute totals or other aggregate functions for each group. The summed totals for each group are output from the stage via an output link.

If you want to aggregate the input data in a number of different ways, you can have several output links, each specifying a different set of properties to define how the input data is grouped and summarized.

When you edit an Aggregator stage, the Aggregator Stage dialog box appears:



This dialog box has three pages:

- **Stage.** Displays the name of the stage you are editing. This page has a **General** page which contains an optional description of the stage and names of before- and after-stage routines. For more details about these routines, see “Before-Stage and After-Stage Subroutines” on page 12-2.

**Note:** You cannot change the name of the stage from this dialog box. For details on how to change stage names, see “Renaming Stages” on page 5-14.

- **Inputs.** Specifies the column definitions for the data input link.
- **Outputs.** Specifies the column definitions for the data output link.

Click **OK** to close this dialog box. Changes are saved when you save the job.

## Before-Stage and After-Stage Subroutines

The **General** page on the **Stage** page contains optional fields that allow you to define routines to use which are executed before or after the stage has processed the data.

- **Before-stage subroutine** and **Input Value.** Contain the name (and value) of a subroutine that is executed before the stage starts to process any data. For example, you can specify a routine that prepares the data before processing starts.
- **After-stage subroutine** and **Input Value.** Contain the name (and value) of a subroutine that is executed after the stage has processed the data. For example, you can specify a routine that sends an electronic message when the stage has finished.

Choose a routine from the drop-down list box. This list box contains all the routines defined as a **Before/After Subroutine** under the **Routines** branch in the Repository. Enter an appropriate value for the routine’s input argument in the **Input Value** field.

If you choose a routine that is defined in the Repository, but which was edited but not compiled, a warning message reminds you to compile the routine when you close the Aggregator Stage dialog box.

If you installed or imported a job, the **Before-stage subroutine** or **After-stage subroutine** field may reference a routine that does not exist on your system. In this case, a warning message appears when you close the Aggregator Stage dialog box.

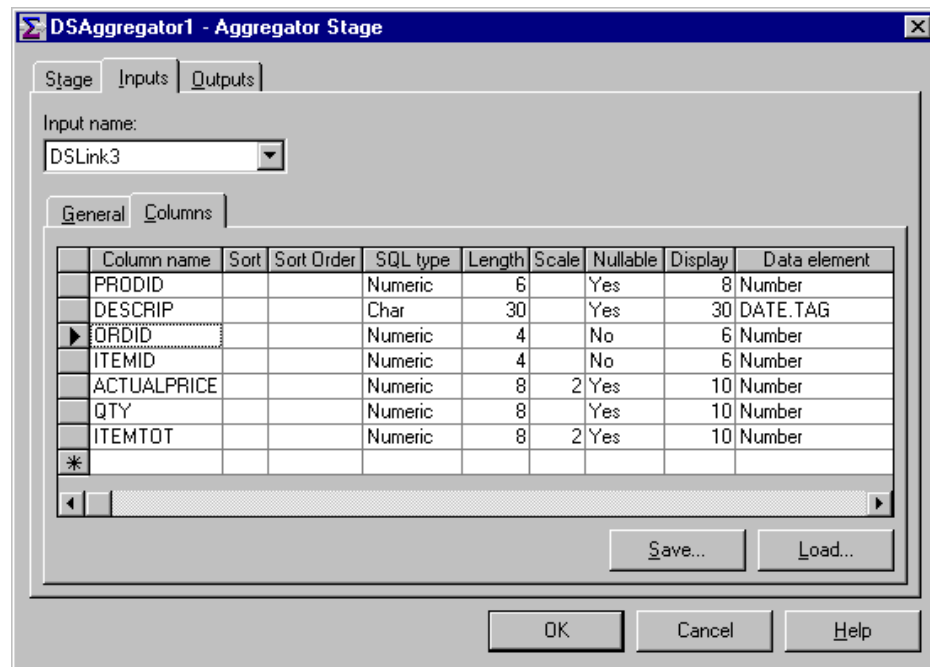


You must install or import the “missing” routine or choose an alternative one to use.

If you edit a job created using Release 1 of DataStage, the **Before-stage subroutine** or **After-stage subroutine** field may contain the name of a routine created at Release 1. When DataStage is upgraded, these routines are identified and automatically renamed. For example, if you used a before-stage subroutine called BeforeSubr, this appears as BeforeSubr\<Rev1> in the **Before-stage subroutine** field. You can continue to use these routines. However, because you could not specify input values for routines at Release 1 of DataStage, the **Input Value** field grays out when you use one of these “old” routines.

## Defining Aggregator Input Data

Data to be aggregated is passed from a previous stage in the job design and into the Aggregator stage via a single input link. The properties of this link and the column definitions of the data are defined on the **Inputs** page in the Aggregator Stage dialog box.



The **Inputs** page has the following field and two pages:

- **Input name.** The name of the input link to the Aggregator stage.
- **General.** Displayed by default. Contains an optional description of the link.
- **Columns.** Contains a grid displaying the column definitions for the data being written to the stage, and an optional sort order.
  - **Column name.** The name of the column.
  - **Sort.** Displays the sort key position of the column, if sorting is enabled. For more information, see “Defining the Input Column Sort Order” on page 12-4.
  - **Sort Order.** Specifies the sort order. This field is blank by default, that is, there is no sort order. Choose **Ascending** for ascending order, **Descending** for descending order, or **Ignore** if you do not want the order to be checked.
  - **SQL type.** The SQL data type.
  - **Length.** The data precision. This is the length for CHAR data and the maximum length for VARCHAR data.
  - **Scale.** The data scale factor.
  - **Nullable.** Specifies whether the column can contain null values.
  - **Display.** The maximum number of characters required to display the column data.
  - **Data element.** The type of data in the column.
  - **Description.** A text description of the column.

For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

## Defining the Input Column Sort Order

When the Aggregator stage collates input data for aggregating, it is stored in memory. If one or more group columns in the input data are sorted, this can greatly improve the way in which the Aggregator stage handles the data.

Sorted input data can be output from an ODBC or UniVerse stage (using an ORDER BY clause in the SQL statement) or a Sequential File stage.

To use sorted input data, you can use the additional column properties in the Input Column dialog box.

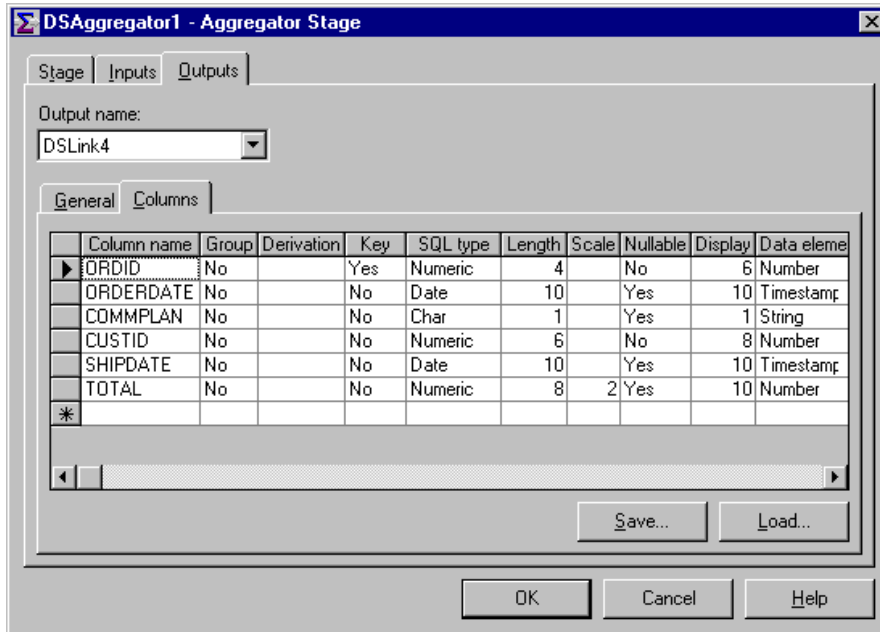
Enter a number in the **Sort** column specifying the position that column has in the sort key. For example, if the input data was sorted on a date then on a product code, the sort key position for the date column would be 1 and the sort key position for the product code column would be 2. A value of 1 always indicates the most significant key. If you do not specify a value in this field, the column is added to the end of the sort key sequence. Once you click **OK**, all the columns are sorted in sequence from the most significant column upward.

Choose the order in which the data is sorted from the **Sort Order** column:

- **Ascending.** This is the default setting. Choose this option if the input data in the specified column is sorted in ascending order. If you choose this option, the DataStage server checks the order at run time.
- **Descending.** Choose this option if the input data in the specified column is sorted in descending order. If you choose this option, the DataStage server checks the order at run time.
- **Ignore.** Do not check order. Choose this option if the sort order used by the input data is not simply ascending or descending order, but uses a more complex sort order. You must take care when choosing this option. At run time the DataStage server does not check the sort order of the data, which may cause erroneous errors. If you choose this option, a warning message appears when you click **OK**. You must acknowledge this message before you can edit other input columns.

## Defining Aggregator Output Data

When you output data from an Aggregator stage, the properties of output links and the column definitions of the data are defined on the **Outputs** page in the Aggregator Stage dialog box.



The **Outputs** page has the following field and two pages:

- **Output name.** The name of the output link. Choose the link to edit from the Output name drop-down list box. This list box displays all the output links from the stage.
- **General.** Displayed by default. Contains an optional description of the link.
- **Columns.** Contains a grid displaying the column definitions for the data being output from the stage. The grid has the following columns:
  - **Column name.** The name of the column.
  - **Group.** Specifies whether to group by the data in the column. Choose **Yes** to group by the column.
  - **Derivation.** Contains an expression specifying how the data is aggregated. This is a complex cell, requiring more than one piece of information. Double-clicking on the cell invokes the Derivation dialog box. For more information, see “Aggregating Data” on page 12-7.
  - **Key.** Indicates whether the column is part of the primary key.

- **SQL type.** The SQL data type.
- **Length.** The data precision. This is the length for CHAR data and the maximum length for VARCHAR data.
- **Scale.** The data scale factor.
- **Nullable.** Specifies whether the column can contain null values.
- **Display.** The maximum number of characters required to display the column data.
- **Data element.** The type of data in the column.
- **Description.** A text description of the column.

For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

## Aggregating Data

The data sources you are extracting data from can contain many thousands of rows of data. For example, the data in a sales database can contain information about each transaction or sale. You could pass all this data into your data warehouse. However, this would mean you would have to search through large volumes of data in the data warehouse before you get the results you need.

If you only want summary information, for example, the total of product A sold since 01/01/96, you can aggregate your data and only pass the summed total to the data warehouse. This reduces the amount of data you store in the data warehouse, speeds up the time taken to find the data you want, and ensures the data warehouse stores data in a format you need.

The Aggregator stage allows you to group by or summarize any columns on any of the output links.

**Note:** Every column output from an Aggregator stage must be either grouped by or summarized.

A group of input data is a set of input rows that share the same values for all the grouped by columns. For example, if your sales database contained information about three different products A, B, and C, you could group by the **Product** column. All the information about product A would be grouped together, as would all the information for products B and C.

By summarizing data, you can perform basic calculations on the values in a particular column. The actions you can perform depend on the SQL data type of the selected column.

For numeric SQL data types you can perform the following actions:

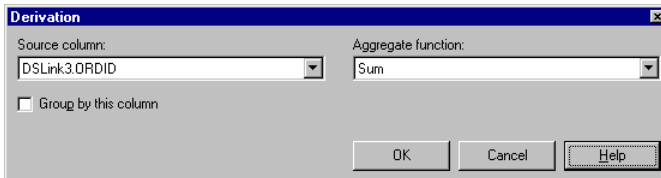
- **Minimum.** Returns the lowest value in the column.
- **Maximum.** Returns the highest value in the column.
- **Count.** Counts the number of values in the column.
- **Sum.** Totals the values in the column.
- **Average.** Averages the values in the column.
- **Standard deviation.** Returns the standard deviation for the values in the column.

For any other SQL data types you can perform the following actions:

- **Minimum.** Returns the lowest value in the column.
- **Maximum.** Returns the highest value in the column.
- **Count.** Counts the number of values in the column.

For example, if you want to know the total number of product A sold, you would sum the values in the **QtySold** column.

To group by or summarize a column, you must edit the **Derivation** column in the Output Column dialog box. Do this by double-clicking the cell to invoke the Derivation dialog box.



The Derivation dialog box contains the following fields and option:

- **Source column.** Contains the name of the column you want to group by or summarize, in the format *linkname.columnname*. You can choose any of the input columns from the drop-down list box.
- **Aggregate function.** Contains the aggregation function to perform. Choose the function you want from the drop-down list box. The default option is **Sum**.

- **Group by this column.** Specifies whether the column will be grouped. This check box is cleared by default.

If you want to group by the column, select the **Group by this column** check box. The aggregate function is automatically set to **(grouped)**, and you cannot select an aggregate function from the drop-down list box.

To use an aggregate function, clear the **Group by this column** check box and select the function you want to use from the Aggregate function drop-down list box.

Click **OK** to save the settings for the column.

## The AGGREGATOR Plug-In

At DataStage Release 1.0, aggregation was performed using the AGGREGATOR plug-in. This plug-in is no longer available and is superseded by the built-in Aggregator stage. During installation, any jobs that used the AGGREGATOR plug-in are converted to use the built-in Aggregator stage instead. You do not need to recompile your job to use the Aggregator stage.





# 13

## Plug-Ins and Plug-In Stages

This chapter describes how to use a plug-in stage in your job design.

You may find that the built-in stage types do not meet all your requirements for data extraction and transformation. If this is the case, you need to obtain a plug-in, which can then be used in a plug-in stage in your job design.

### Plug-Ins

Plug-ins are written to perform specific tasks that the built-in stages do not support, for example:

- Custom aggregations
- Control of external devices (for example, tape drives)
- Access to external programs

Two plug-ins are automatically installed with DataStage:

- **BCPLoad.** The BCPLoad plug-in bulk loads data into a single table in a Microsoft SQL Server (Release 6 or 6.5) or Sybase (System 10 or 11) database. For more information about this plug-in, see Chapter 14, “BCPLoad Stages.”
- **Orabulk.** The Orabulk plug-in generates control and data files for bulk loading into a single table on an Oracle target database. The files are suitable for loading into the target database using the Oracle command *sqlldr*. For more information about this plug-in, see Chapter 15, “Orabulk Stages.”

Other plug-ins are supplied with DataStage, but must be explicitly installed. These plug-ins are located on the DataStage installation CD and are installed using the

Package Installer, which is described in *DataStage Administrator's Guide*. Some plug-ins have a custom GUI, which is also installed. The available plug-ins are listed in the *Read Me* file in the plug-in directory on the CD (*.\Packages*). If the plug-in you require is not listed, contact Ardent to see if one is likely to become available. Alternatively, you can write your own plug-in.

A plug-in consists of a set of routines that access external databases and/or perform complex programming. You must have a thorough knowledge of C to design and develop a plug-in.

To write your own plug-in:

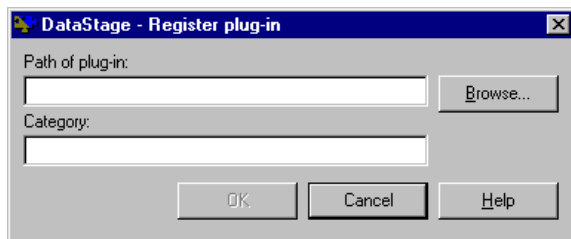
1. Assess the purpose of the plug-in. You need to determine what the plug-in must do in terms of data extraction or data conversion. Check that Ardent does not already have an available plug-in for this purpose.
2. Develop the routines using the DataStage C plug-in Stage Interface. These routines are used to extract, aggregate, or transform data when the job is run. If you need help writing the plug-in routines required to process your data, contact your local Ardent Customer Support Center.
3. Register the plug-in with the DataStage Manager. It is recommended that your plug-in dynamically registers itself. Alternatively, you can register the plug-in manually.

DataStage has a generic Stage dialog box which can be used by plug-ins, but it is also possible to define your own GUI for a plug-in and install that in DataStage.

## Manually Registering a Plug-In Definition

To register a plug-in manually in the DataStage Manager:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools ► Register Plug-in...** . The Register plug-in dialog box appears:



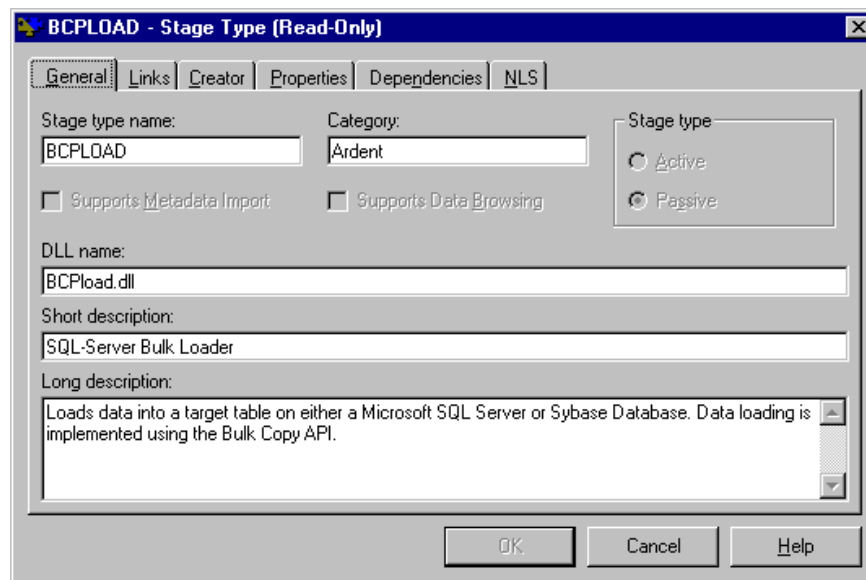
3. Enter the path and name of the plug-in DLL in the **Path of plug-in** field.
4. Specify where the plug-in will be stored in the Repository by entering the category name in the **Category** field.
5. Click **OK** to register the plug-in definition and close the dialog box.

## Viewing Plug-In Definition Details

Once a plug-in has been registered with the Manager, you can view its details in the Stage Type dialog box. To do this, click the **Plug-in** icon in the Manager display area and do one of the following:

- Choose **File ► Properties...**
- Choose **Properties...** from the shortcut menu.
- Double-click the plug-in in the display area.
- Click the **Properties** icon on the toolbar.

The plug-in definition is read-only; you cannot change any of the details. The only exception to this is a plug-in that you added under an earlier version of DataStage.

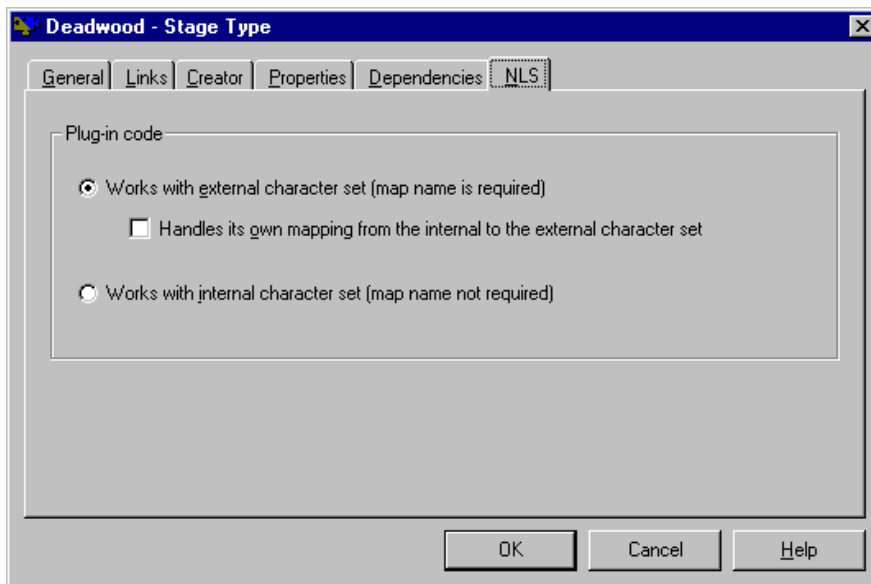


This dialog box has up to six pages:

- **General.** Contains the name and type of the plug-in and optional short and long descriptions. This page also contains the name and path of the plug-in DLL, and specifies whether the plug-in supports meta data import and/or data browsing.
- **Links.** Contains information about what type of links can connect to the plug-in stage as inputs and the maximum and minimum number of links allowed.
- **Creator.** Displays information about the creator and the version of the plug-in.
- **Properties.** Specifies properties for the stage and the input and output links.
- **Dependencies.** Specifies the dependent DLLs.
- **NLS.** Specifies how character set mapping is carried out by the plug-in. For more information, see “Specifying Character Set Mapping.”

### Specifying Character Set Mapping

If you want your plug-in to use data in character sets other than ASCII, you must specify how the plug-in handles character set mapping from the NLS page.



Click **Works with external character set** to specify that the data requires mapping:

- If the plug-in performs its own mapping, you must also select **Handles its own mapping**.
- If the plug-in does not perform the character set mapping, you must specify the map to be used from the plug-in stage dialog box when you use the plug-in in a job design.

If the plug-in does not require any character set mapping, select **Works with internal character set**.

## Removing a Registered Plug-In

To remove a plug-in that you have previously registered:

1. Select the plug-in from the DataStage Manager display area.
2. Choose **Tools ► Unregister Plug-in**. The plug-in is removed.

## Packaging a Plug-In

If you have written a plug-in that you want to distribute to users on other DataStage systems, you need to package it. For details on how to package a plug-in for deployment, see Chapter 19, “Importing, Exporting, and Packaging Jobs.”

## Using a Plug-In

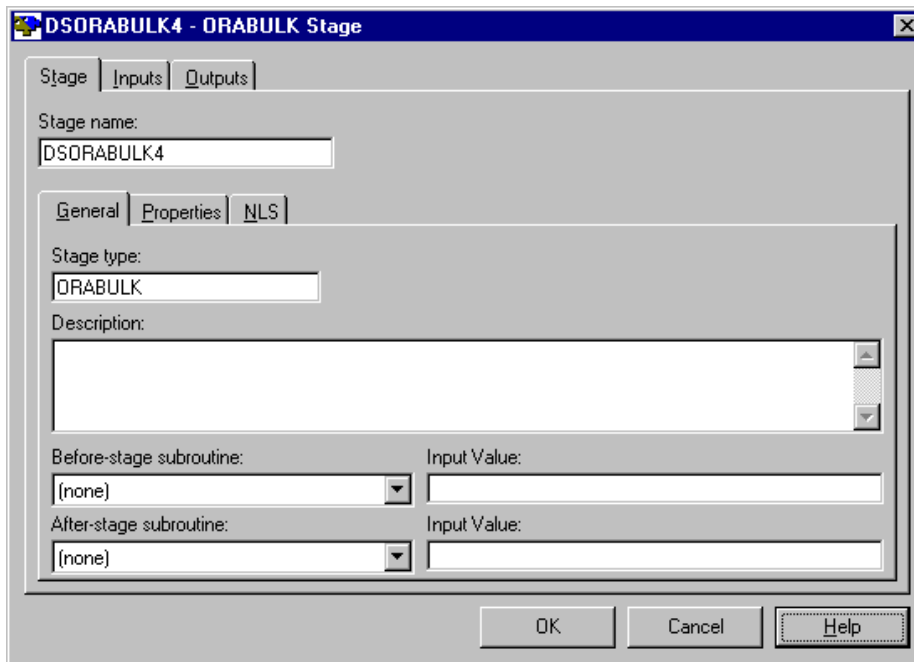
You can use a plug-in by inserting a plug-in stage in your job design. Plug-in stages are used the same way as built-in stages. However, when you insert a plug-in stage, you are prompted to choose the plug-in to use. For details of how to include a plug-in stage in your job design, see “Adding Plug-In Stages” on page 5-13.

The plug-in you choose determines the function and properties of the plug-in stage. When you have chosen a plug-in, you can edit the stage to define the data flowing into, through, or from it.

## Plug-In Stages

A plug-in stage is the way in which you use plug-ins in a job. The plug-in chosen when you insert a plug-in stage determines the behavior of the stage, how it handles data, and the number of input or output links.

All plug-in stages that use the generic DataStage Stage dialog box are edited in the same way. When you edit a plug-in stage, the Stage dialog box appears. The title of the Stage dialog box includes the stage name.



The Stage dialog box can have up to three pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** Displays the name of the stage and contains up to three pages:
  - **General.** Displayed by default. Specifies the plug-in to use and contains an optional description of the stage. Also contains the names of cataloged routines, if you are editing an active stage. For more details about these routines, see “Before-Stage and After-Stage Subroutines” on page 13-7.
  - **Properties.** Displays the stage properties. See “Editing Properties” on page 13-10 for information on how to edit property values.

- **NLS.** If NLS is enabled, allows you to specify a character set map for the stage. See “Defining Character Set Maps” on page 13-12.

**Note:** You cannot change the name of the stage from this dialog box. For details on how to change stage names, see “Renaming Stages” on page 5-14.

- **Inputs.** Contains information about the column definitions for each data input link and specifies the input link properties. For more information, see “Defining Plug-In Input Data” on page 13-8.
- **Outputs.** Contains information about the column definitions for each data output link and specifies the output link properties. For more information, see “Defining Plug-In Output Data” on page 13-9.

## Before-Stage and After-Stage Subroutines

If you are editing an active type of plug-in stage, the **General** page on the **Stage** page contains additional fields. These optional fields allow you to define routines to use which are executed before or after the stage has processed the data:

- **Before-stage subroutine and Input Value.** Contain the name (and value) of a subroutine that is executed before the stage starts to process any data. For example, you can specify a routine that prepares the data before processing starts.
- **After-stage subroutine and Input Value.** Contain the name (and value) of a subroutine that is executed after the stage has processed the data. For example, you can specify a routine that sends an electronic message when the stage has finished.

Choose a routine from the drop-down list box. This list box contains all the built routines defined as a **Before/After Subroutine** under the **Routines** branch in the Repository. Enter an appropriate value for the routine’s input argument in the **Input Value** field.

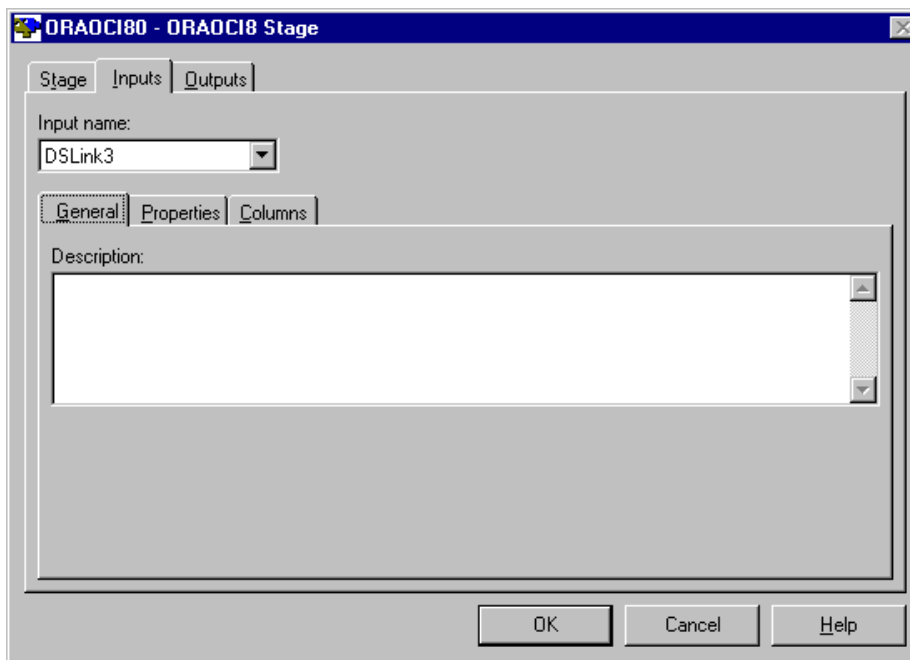
If you choose a routine that is defined in the Repository, but which was edited but not compiled, a warning message reminds you to compile the routine when you close the Stage dialog box.

If you installed or imported a job, the **Before-stage subroutine** or **After-stage subroutine** field may reference a routine that does not exist on your system. In this case, a warning message appears when you close the Stage dialog box. You must install or import the “missing” routine or choose an alternative one to use.

If you edit a job created using Release 1 of DataStage, the **Before-stage subroutine** or **After-stage subroutine** field may contain the name of a routine created at Release 1. When DataStage is upgraded, these routines are identified and automatically renamed. For example, if you used a before-stage subroutine called BeforeSubr, this appears as BeforeSubr\<Rev1> in the **Before-stage subroutine** field. You can continue to use these routines. However, because you could not specify input values for routines at Release 1 of DataStage, the **Input Value** field grays out when you use one of these “old” routines.

## Defining Plug-In Input Data

When a plug-in stage accepts data from another stage in the job design, the stage has an input link. The properties of this link and the column definitions of the data are defined on the **Inputs** page in the Stage dialog box.



The **Inputs** page contains the following field and up to three pages:

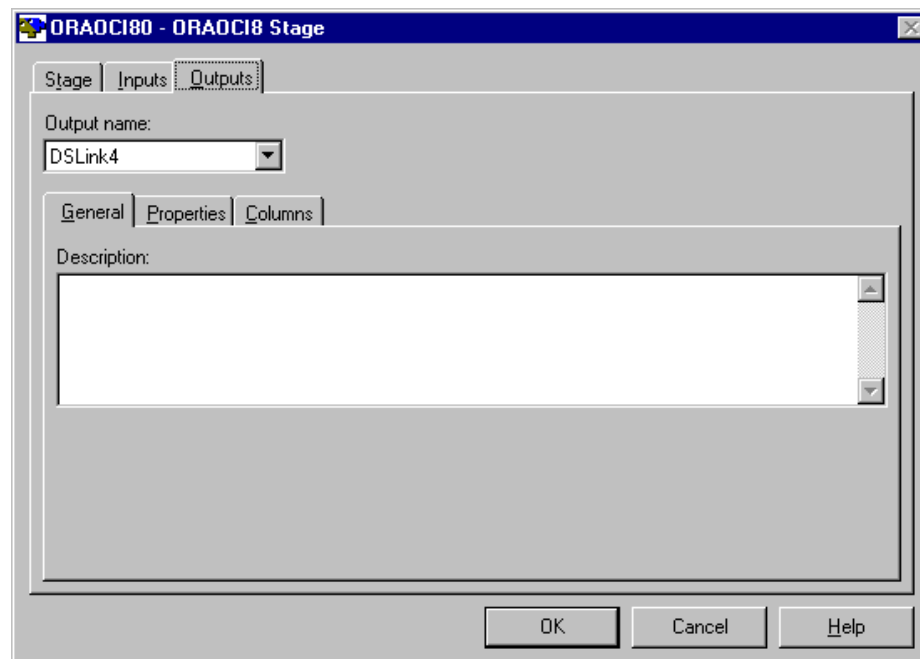
- **Input name.** The name of the input link. Choose the link you want to edit from the Input name drop-down list box. This list box displays all the input links to the plug-in stage.



- **General.** Displayed by default. Contains an optional description of the link.
- **Properties.** Displays the input link properties. This page is displayed only if input properties have been defined for the chosen plug-in. See “Editing Properties” on page 13-10 for information on how to change the property values.
- **Columns.** Contains the column definitions for the data being written to the stage. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

## Defining Plug-In Output Data

If the plug-in stage represents a data source you are extracting data from, the stage has an output link. The properties of the output link and the column definitions of the data are defined on the **Outputs** page in the Stage dialog box.



The **Outputs** page has the following field and up to three pages:

- **Output name.** The name of the output link. Choose the link to edit from the Output name drop-down list box. This list box displays all the output links from the stage.
- **General.** Displayed by default. Contains an optional description of the link.
- **Properties.** Displays the output link properties. This page is displayed only if output properties have been defined for the chosen plug-in. See “Editing Properties” on page 13-10 for information on how to change the property values.
- **Columns.** Contains the column definitions for the data being output from the stage. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

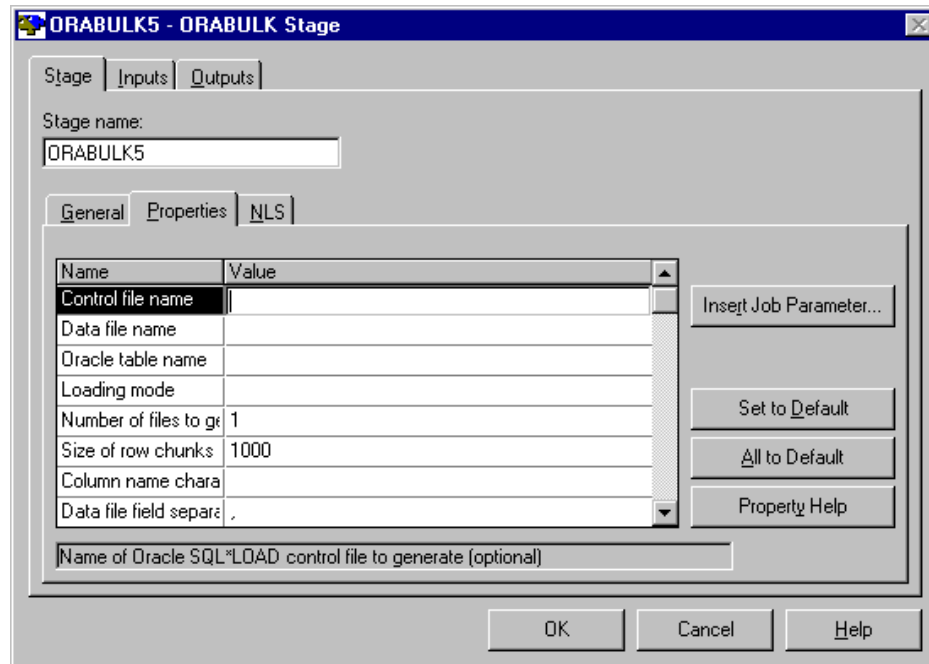
## Editing Properties

When a plug-in is created, properties are defined for the stage and the input and output links. The name and meaning of each property is defined by the creator of the plug-in. For details on how to define plug-ins, see “Manually Registering a Plug-In Definition” on page 13-2.

Each property has a default value and optional help text. These properties and default values are automatically assigned to a plug-in stage when you specify the plug-in to use.

You can view the properties and enter more appropriate values using the **Properties** page on the **Stage**, **Inputs**, or **Outputs** page. The **Properties** page contains the same fields and columns.

The following screen shows the **Properties** page on the **Stage** page:



The properties are displayed in a grid which has the following columns:

- **Name.** Displays the caption text used to describe the property.
- **Value.** Displays the current setting for the property value. If the property has not been edited, this is the default value for the property.

There are four buttons on this page:

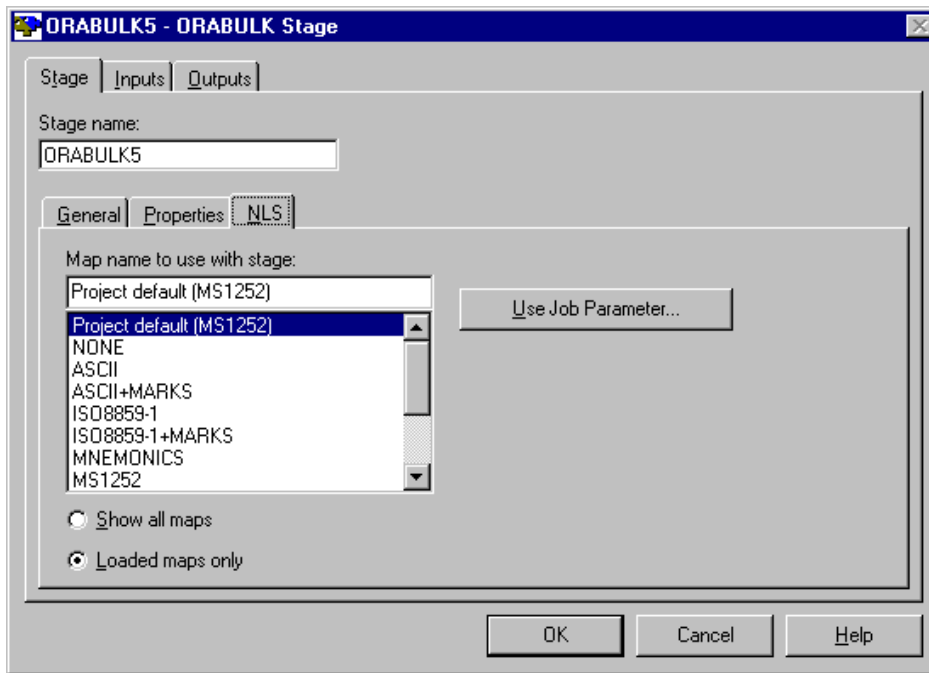
- **Insert Job Parameter...** . Allows you to insert a job parameter as the value for a chosen property. When you click this button, a list appears displaying the currently defined job parameters. Choose one from the list. If you cannot see the job parameter you want, click (**New...**) to define a new one. The Job Parameters dialog box appears. For more information about this dialog box and how to define a job parameter, see “Parameter Types” on page 5-34. You can also insert a job parameter using the **F9** key.
- **Set to Default.** Sets the value for the chosen property to the default value. This button is only active when you choose a property to edit.
- **All to Default.** Sets the values for all properties to the default values.

- **Property Help.** Displays the help text supplied by the creator of the plug-in, if there is any. This button is only active when you choose a property to edit.

You can edit the value for any properties listed in the grid. Click **OK** to save the settings and to close the Stage dialog box.

## Defining Character Set Maps

You can define a character set map for a plug-in stage using the **NLS** page of the plug-in stage dialog box.



The default character set map (defined for the project or the job) can be changed by selecting a map name from the list.

# 14

## BCPLoad Stages

The BCPLoad stage is a passive plug-in stage provided with Informix DataStage. It bulk loads data into a single table in a Microsoft SQL Server (Release 6.0 or 6.5) or Sybase (System 10 or 11) database. The files are loaded into the target database using the bulk copy API.

By default, the BCPLoad stage is configured to bulk load data into a Microsoft SQL Server. You can configure the BCPLoad stage properties to bulk load data into a Sybase SQL Server table using the Sybase DBLIB or CTLIB client libraries.

**Note:** The client libraries used by the BCPLoad stage are not supplied as part of DataStage. You must obtain these libraries from your DBMS vendor and ensure they are installed and configured on your system before attempting to use the BCPLoad stage.

There is one input link to this stage which provides a sequence of rows to load into the SQL Server or Sybase database table. The meta data for each input column determines how it is loaded. There are no output links from this stage type.

### Overview

Microsoft SQL Server and Sybase have a utility called BCP (Bulk Copy Program). This command line utility copies SQL Server data to or from an operating system file in a user-specified format. BCP uses the bulk copy API in the SQL Server client libraries.

By using BCP, you can load large volumes of data into a table without recording each insert in a log file. You can run BCP manually from a command line using command line options (switches). A format (*.fmt*) file is created which is used to load the data into the database.

The BCPLoad stage uses the same API that BCP does, but loads data directly without the need for a format file. The command line switches are set using stage properties.

Because this version of the BCPLoad stage supports both Microsoft SQL Server and Sybase, only BCP switches common to both servers have been included as stage properties. The following command line switches are not supported for Microsoft SQL Server:

- -T, trusted connection
- -q, quote identifiers

The following command line switches are not supported for Sybase:

- -I, interface file
- -J, the client character set
- -q, the data character set

For more information about the BCP switches that can be set, see “Stage Properties” on page 14-4.

The BCPLoad stage does not support the loading of native data files.

## Before You Start

Before you can use the BCPLoad stage you must:

- Install and configure the SQL Server or Sybase client software. The BCPLoad stage uses the BCP API in the DBLIB/CTLIB and NetLIB client libraries. You must ensure that these components are installed on the DataStage server which is acting as a client to the SQL Server DBMS. See the documentation supplied with your DBMS for more details.
- Use one of the client tools (for example, ISQLW in the case of Microsoft SQL Server or WISQL32 for Sybase) to ensure that the connectivity between the DataStage server and the SQL Server host is operational.
- Create the table in the database on the SQL Server.
- Configure your database to use the fast copy (bulk load) option. By using this option, the data is loaded without each insert being recorded in a log file. If you do not specify this setting, all transactions are logged, slowing down the rate at which data is loaded. The fast copy option can be switched on by a stored procedure. For more information about using stored procedures, see “Using Stored Procedures” on page 14-7.

## Table Definitions

You can import the table definition from the table in your database on the SQL Server by using the meta data import option in the DataStage Manager. The table definition is imported via an ODBC connection to the Server. You can then load this table definition into the stage connected to the BCPLoad stage.

For more information about importing meta data, see “Importing a Table Definition” on page 4-11.

## SQL Data Types

The following SQL Server data types are supported by the BCPLoad stage:

- Bit
- Char
- DateTime
- Decimal
- Float
- Integer
- Money
- Numeric
- Real
- SmallDateTime
- SmallInt
- SmallMoney
- TinyInt
- VarChar

When you import meta data from your database table, these data types are mapped to appropriate SQL data types by the ODBC driver. You can view the data types used in the table definition with the DataStage Manager, or when you edit a stage in your job design.

The following SQL Server data types are not supported by the BCPLoad stage:

- Binary
- VarBinary
- Image
- Text (large text which is a binary type)

## The BCPLoad Plug-In Definition

When the BCPLoad plug-in is installed, it is stored under **Stage Types** ► **Ardent** in the Repository. You can view the definition of this plug-in using the DataStage Manager, but you cannot edit it. When you choose to view the definition, the Stage Type dialog box appears. This dialog box has up to six pages.

- **General.** Specifies where the BCPLoad plug-in is installed and contains a description of the stage.
- **Links.** Specifies the type of input links which can connect to a BCPLoad stage and gives the maximum and minimum links allowed.
- **Creator.** Contains information about the creator of this stage.
- **Properties.** Contains the stage properties for the BCPLoad stage with suitable default settings. For a list of the properties, see “Stage Properties” on page 14-4.
- **Dependencies.** Contains the name of any DLLs used by the plug-in.
- **NLS.** Specifies how character set mapping is carried out for the stage.

## Stage Properties

The following properties are defined for the BCPLoad stage. You can view them on the **Properties** page in the Stage Type dialog box and change them when you edit the stage instance.

- **SQL-Server Name.** The name of the SQL Server to connect to. This property corresponds to the BCP -S switch. This property is optional and has no default value. If you leave this property blank in the stage instance, the stage assumes the SQL Server resides on the same machine as the DataStage Server.
- **User ID.** The logon name of the SQL user. This property corresponds to the BCP -U option. There is no default value for this property. You must enter the name of the user when you edit the stage instance.
- **Password.** The password of the SQL user. This property corresponds to the BCP -P option. There is no default value for this property. You must enter a password when you edit the stage instance.
- **Database Name.** The name of the database to use on the SQL Server. This property has no default value. You must enter the name of the database when you edit the stage instance.



- **Table Name.** The name of the table to load data into. This property has no default value. You must enter the name of the table to use when you edit the stage instance.
- **Before Load Stored Procedure.** The name of a stored procedure that is executed before the database table is loaded. This property is optional and has no default value. For more information about using a before-load stored procedure, see “Using Stored Procedures” on page 14-7.
- **After Load Stored Procedure.** The name of a stored procedure that is executed after the database table is loaded. This property is optional and has no default value. For more information about using an after-load stored procedure, see “Using Stored Procedures” on page 14-7.
- **Batch Size.** The number of rows to include in the BCP batch. This property corresponds to the BCP -b option. The default setting for this property is 0, that is, all the data rows are treated in one batch. If an error occurs, all rows are rolled back.
- **Packet Size.** The number of bytes per network packet sent to and from the server. The default value is 4096. When you edit the stage instance, you can enter any number from 512 through 65535.
- **Use Source Identity Data.** This property corresponds to the BCP /E switch. Setting this property tells the SQL Server to use the identity values that are passed to it by the BCPLoad stage, to populate the corresponding identity column in the SQL Server table.
- **Date Format.** This property provides a workaround to the problem that Microsoft SQL Server has with dates in YMD format. If your target table has a date column and your data has dates in YMD format, a conversion is required for the date to load successfully. By setting this property to ymd, dates are automatically converted during loading to a format that Microsoft SQL Server accepts.
- **Client Library.** The type of client library to use. The default setting is MSDBLIB (the Microsoft DBLibrary). Other valid settings are SYBDBLIB for the Sybase DBLibrary and SYBCTLIB for the Sybase CTLibrary.

## Using the BCPLoad Stage

The BCPLoad plug-in is preinstalled and can be used immediately. To use this plug-in, you must insert a plug-in stage in your job design:

1. Start the DataStage Designer and open your job design.

2. Click the **Plug-in Stage** icon on the toolbar. The Plug-in Stage dialog box appears listing all the plug-ins installed on the DataStage Server.
3. Choose **BCPLoad** from the list box and click **OK**.
4. Click in the Diagram window where you want to position the stage.

Link an output from a relevant stage in the job design to the input of the BCPLoad stage.

## Editing the BCPLoad Stage

The BCPLoad stage is edited in the same way as any other DataStage stage. When you edit this stage, the BCPLoad Stage dialog box appears. This dialog box has two pages:

- **Stage.** Contains the name of the stage you are editing. This page has up to three pages:
  - **General.** Contains an optional description of the stage and the stage type (BCPLoad).
  - **Properties.** Contains the stage properties and their current values. You can edit the settings for the stage properties or specify job parameters. For more information about using job parameters on this page, see “Using Job Parameters” on page 14-6.
  - **NLS.** If NLS is enabled and you do not want to use the project default character set map, you can select an alternative character set map from this page.
- **Inputs.** Contains the name of the input link. This page has two pages:
  - **General.** Contains an optional description of the link.
  - **Columns.** Contains the column definitions for the data you are loading into your database table. The column definitions on this page are specified by the meta data defined on the output link of the connected stage. For a description of how to enter and edit column definitions, see “Specifying Column Definitions” on page 5-18.

## Using Job Parameters

You can use job parameters for any of the stage properties in the BCPLoad stage. For a description of each of these properties, see “Stage Properties” on page 14-4.

When you validate or run the job, you are prompted to enter suitable values for the properties.

To use a job parameter:

1. Define the job parameter. For more details about defining job parameters, see “Parameter Types” on page 5-34.
2. Click the **Properties** tab on the **Stage** page in the BCPLoad Stage dialog box. The **Properties** page appears at the front of the **Stage** page.
3. Click the **Value** cell for the property you want to edit.
4. Click **Insert Job Parameter...** or press **F9**. A list appears displaying the currently defined job parameters. Choose one from the list, or choose **New** to invoke the Job Properties dialog box to define a new parameter.
5. Click **OK** to close the BCPLoad Stage dialog box. Any changes are saved when you save your job design.

## Defining Character Set Maps

You can define a character set map for the stage instance using the **NLS** page of the Stage Type dialog box. You can choose a specific character set map from the list or accept the default setting for the whole project.

**Note:** The list contains all the character set maps that are loaded and ready for use. You can view other maps that are supplied with DataStage by clicking **Show all maps**, but these maps cannot be used unless they are loaded using the DataStage Administrator. For more information, see *DataStage Administrator's Guide*.

## Using Stored Procedures

You can enter the name of a stored procedure to execute before or after loading the database. Before-load stored procedures can be used to perform tasks such as dropping indexes and turning on the database bulk copy option. After-load stored procedures can be used to turn off the bulk copy option and recreate any indexes. For a detailed description of how to write a stored procedure, see the SQL Server documentation.

The stored procedure name is entered as the value for the **Before Load Stored Procedure** or **After Load Stored Procedure** stage property. As well as entering the name of a stored procedure you can also include parameter values. To enter

parameters for the stored procedure, use the following format in the **Value** field on the **Properties** page on the **Stage** page:

*procedurename P1, P2, P3, ..., Pn*

*procedurename* is the name of the stored procedure. *P1...Pn* are parameter values, in the order expected by the stored procedure. Note that string values must be quoted.

If you want to return messages from a stored procedure and write them to the job log file, you can use the output parameters DSSeverity and DSMessage. These parameters return messages to the job log file with an appropriate severity. The type of message written to the job log file depends on the value returned by the DSSeverity parameter:

- Return value of 0. Nothing is written.
- Return value of 1. An informational message is written.
- Return value of 2. A warning message is written.
- Return value of 3. A fatal message is written. The DataStage job aborts and any return values from the stored procedure other than the DataStage expected output parameters, are ignored.

For more information about the job log file, see *DataStage Operator's Guide*.

The following example of a before-load stored procedure is supplied as part of the BCPLoad demo SQL script, *BCPDemo.sql*. For more information about the BCPLoad demo, see "The BCPLoad Demo" on page 14-11.

This stored procedure demonstrates the use of the output parameters DSSeverity and DSMessage:

```
create proc DemoBeforeSP
    @lReplace bit,
    @DSSeverity int output,
    @DSMessage varchar(255) = "" output
as
/* Remove the following three lines if running on Sybase */
declare @sSetDBOption varchar(255)
select @sSetDBOption = 'sp_dboption' + DB_NAME() + ", 'select
↳ into/bulkcopy', TRUE"

exec (@sSetDBOption)

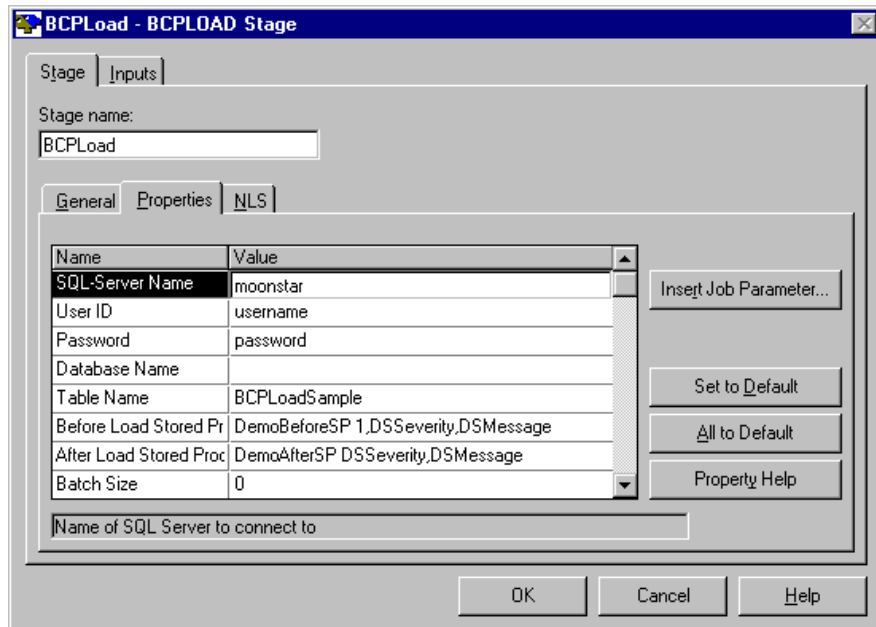
if @lReplace = 1
begin
    truncate table BCPLoadSample
end
```

```

if @@ERROR = 0
begin
    select @DSMessage = "Before SP completed: "
    if @lReplace = 1
    begin
        select @DSMessage = @DSMessage + "replacing existing data"
    end
    else
    begin
        select @DSMessage = @DSMessage + "appending data"
    end
    end
    select @DSSeverity = 1                /* INFO */
end
else
begin
    select @DSMessage = "Before SP failed"
    select @DSSeverity = 2                /* WARNING */
end
end
GO

```

To use this stored procedure, enter **DemoBeforeSP 1,DSSeverity,DSMessage** as the value for the **Before Load Stored Procedure** property when you edit the stage instance:



To use existing stored procedures, enter the name of the stored procedure and appropriate parameter values as the value for the **Before Load Stored Procedure** or **After Load Stored Procedure** property.

For example, say your stored procedure includes the following:

```
create proc sp_TrustyDebuggedProcedure
    @strTableName char(30),
    @strSurname char(30),
    @iRowCount int = 0 output
as
...
...
```

If you want to use this procedure as a before-load procedure, you would enter **sp\_TrustyDebuggedProcedure "Table1", "Smith"** in the **Value** field for the **Before Load Stored Procedure** property. "Table1" and "Smith" are passed in as **strTableName** and **strSurname** respectively.

If you want to modify an existing stored procedure to return a severity warning and an error message, the create procedure needs to be modified to include the two output parameters **DSSeverity** and **DSMessage**. In the earlier example, the create procedure would become:

```
create proc sp_TrustyDebuggedProcedure
    @strTableName char(30),
    @strSurname char(30),
    @iRowCount int = 0 output,
    @DSSeverity int output,
    @DSMessage varchar(255) = "" output
as
...
.../* Somewhere in the procedure set appropriate values for DSSeverity
and DSMessage*/
```

In this case, you would enter the following in the **Value** field for the **Before Load Stored Procedure**:

```
sp_TrustyDebuggedProcedure "Table1", "Smith", 0, DSSeverity, DSMessage
```

You can include job parameters to represent the value of a stored procedure parameter. To use job parameters in the earlier example, you would enter the following in the **Value** field for the **Before Load Stored Procedure**:

```
sp_TrustyDebuggedProcedure #Table#, #Name#, 0, DSSeverity, DSMessage
```

*Table* and *Name* are the names of two defined job parameters.

## The BCPLoad Demo

As well as the BCPLoad plug-in itself and example stored procedures, a BCPLoad demo is also supplied. The demo is designed to show how the BCPLoad stage works, using a simple three-stage job design.

The demo comprises three files which are found in the \BCPDEMO directory on the DataStage CD:

- *BCPDemo.txt*. An exported job with associated table definitions.
- *BCPDemo.sql*. An SQL script that creates a demo target table in a chosen database on your SQL Server.
- *BCPDemo.csv*. A sequential file containing 5000 rows of sample data.

The BCPLoad demo job has three stages: a Sequential File stage (for the sample data), a Transformer stage for mapping column names from the source data to the column names in the target table, and a BCPLoad stage for bulk loading the data into the target table.

You can work through the demo step by step, by following these instructions:

1. Using the client configuration tools supplied by your SQL Server vendor, ensure that you have connectivity from your DataStage Server machine to your target SQL Server via DBLIB.
2. Using an appropriate SQL Server tool (for example, ISQL, ISQLW, WISQL32, or Enterprise Manager), load the *BCPDemo.sql* script.

**Note:** *For Sybase users.* If you are loading the sample into a Sybase database, you must comment out the marked section in the DemoBeforeSP and DemoAfterSP stored procedures. You must be logged into the Master database in order to set the **select into/bulkcopy** database option. Once you have run the supplied script, manually set the bulk copy option for the selected database, otherwise the BCPLoad stage will use the BCP slow copy mode.

The script creates a table called BCPLoadSample as follows:

```
CREATE TABLE BCPLoadSample(  
    CompanyID          numeric(18,0) IDENTITY NOT NULL,  
    CompanyName        char (30) NULL,  
    Address             varchar (255) NULL,  
    State              char (30) NULL,  
    StillTrading       bit NOT NULL,  
    SummaryDate        smalldatetime NULL,  
    Time Updated       datetime NULL,
```

```

AnnualProfit          money, NULL,
CompanyFees           smallmoney NULL,
NumberOfStaff         smallint NULL,
NumberOfVPs          tinyint NULL,
TaxRate              numeric (18, 4) NULL,
BestSellerUnitsSold  real NULL,
TotalUnitsSold       float NULL

```

)

This table includes all the data types supported by the BCPLoad stage.

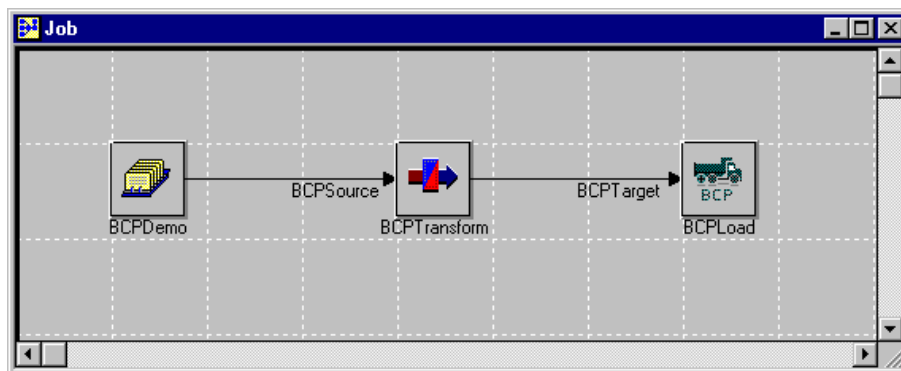
3. Import the *BCPDemo.txt* file using **Tools > Import > DataStage Components** in the DataStage Manager. For more details about importing job components, see Chapter 19, “Importing, Exporting, and Packaging Jobs.”

During the import, the job is imported into your chosen project and two new table definitions are created:

- **BCPLoad.** Defines the column definitions for the output link of the Transformer stage, which is the input to the target table.
- **Sequential.** Defines the column definitions for the source data in the Sequential File stage.

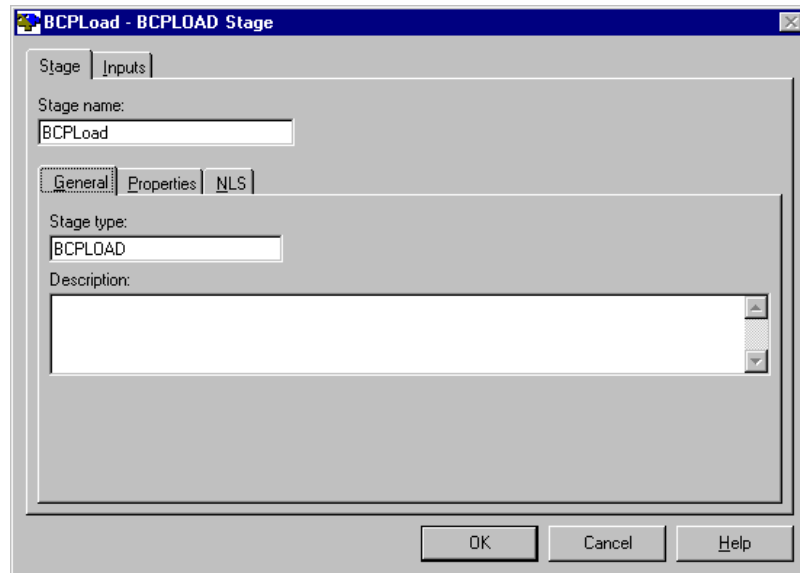
These are stored under **Table Definitions > BCPDemo** in the Repository.

4. Copy the *BCPDemo.csv* sample data file to the project directory containing the imported job. For example, if you were in the project **dstage** when you imported the job, copy the sample data file to the directory `\Ardent\DataStage\dstage`.
5. Start the DataStage Designer and open the BCPDemo job. The following job design appears in the Job window:

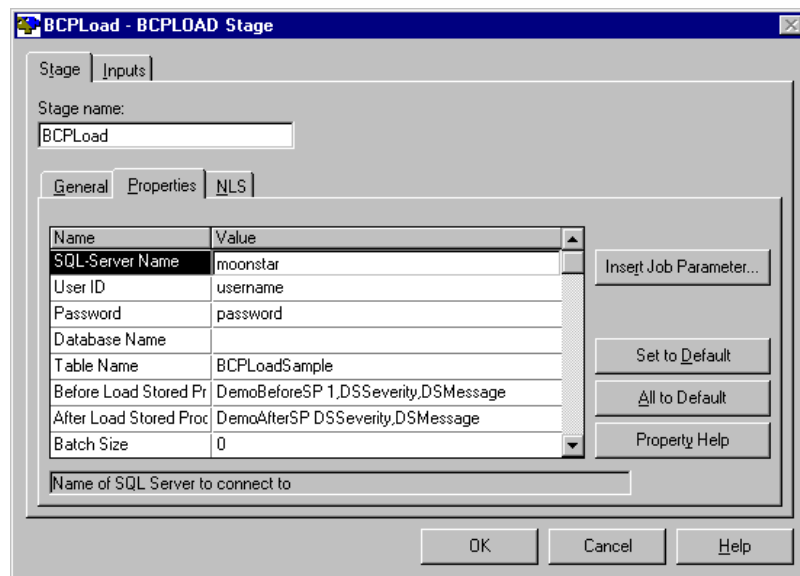




6. Edit the BCPLoad stage. The BCPLoad Stage dialog box appears:



7. Click the **Properties** tab. The **Properties** page appears at the front:



8. Edit the following properties to set appropriate values:
  - **SQL-Server Name**
  - **User ID**
  - **Password** (if required)
  - **Database Name**
  - **Table Name**
9. Click **OK** to close the BCPLoad Stage dialog box.
10. Recompile the job to use the new settings.
11. Start the DataStage Director and run the job. You can use a Monitor window to monitor the rows being written. For more information, see *DataStage Operator's Guide*.

# 15

## Orabulk Stages

The Orabulk stage is a plug-in stage provided with Informix DataStage. It generates control and data files for bulk loading into a single table on an Oracle target database. The files are suitable for loading into the target database using the Oracle command *sqlldr*.

One input link provides a sequence of rows to load into an Oracle table. The meta data for each input column determines how it is loaded. One optional output link provides a copy of all input rows to allow easy combination of this stage with other stages.

### Using the Orabulk Stage

The Orabulk plug-in is preinstalled and can be used immediately. To use this plug-in, you must insert a plug-in stage in your job design:

1. Start the DataStage Designer and open your job design.
2. Click the **Plug-in Stage** icon on the toolbar. The Plug-in Stage dialog box appears with a list of the plug-ins installed on the DataStage server.
3. Choose **Orabulk** from the list box and click **OK**.
4. Click in the Diagram window where you want to position the stage.

Link an output from the relevant stage to the input of the Orabulk stage. The output should deliver a sequence of rows. To store these rows or to process them further, link the output of this stage to an input of another stage. Otherwise, leave the output of this stage unconnected.

## Renaming Columns and Converting Values

You cannot rename columns or convert values during an Orabulk stage. You must do this in an earlier stage. The column names of the input rows must match the column names in the target Oracle table. See the description of the NAMEMAP stage property in “Specifying the Stage Properties” on page 15-3 for a way to map characters in a column name. The data types of the columns should also correspond.

## Integrity Constraints

Values for columns should satisfy any integrity constraints that will be active in the Oracle database when the data is loaded, for example, values that must be nonnull, or unique, or must already exist in other tables. Otherwise, some rows may be rejected when the *sqlldr* command is run.

## Cleaning and Validating Data

The Orabulk stage does not *validate* data values, but it does clean up some data values to match the length, precision, or scale of the data type. For example, CHAR and VARCHAR values are truncated on the right. DECIMAL and NUMERIC values are rounded to their specified scale. If you want to clean data to ensure that it matches the integrity constraints of the target table, you must do so in an earlier stage.

## Specifying the Stage Properties

You can specify the following properties in addition to the standard stage properties:

Property	Description						
Control file name	<p>A local pathname for the Oracle <i>sqlldr</i> control file.</p> <ul style="list-style-type: none"><li>• If no value is specified, the control information is output at the start of the data file. If there is more than one data file, each data file gets a copy of the same control information.</li><li>• If a pathname is specified, a single control file is generated which can be used with any of the data files generated. If the pathname has no suffix, <i>sqlldr</i> adds the suffix <i>.ctl</i> to the pathname.</li></ul> <p>If a control file already exists, it is truncated before processing; if a control file does not exist, it is created.</p>						
Data file name	<p>A local pathname for the Oracle <i>sqlldr</i> data file. If the pathname has no suffix, <i>sqlldr</i> adds the suffix <i>.dat</i> to the pathname. If there is more than one data file, the files are sequentially numbered, for example, <i>datafile1</i>, <i>datafile2</i>, and so on. You can specify a position for the numbering by including a % character in the file name. For example, the value <i>data%file</i> produces files named <i>data1file</i>, <i>data2file</i>, and so on. If a data file already exists, it is truncated before processing; if a data file does not exist, it is created.</p>						
Oracle table name	<p>The name of the target table that the files are loaded into on the Oracle database. This value must be identical to the target table name.</p>						
Loading mode	<p>One of the following values:</p> <table><tbody><tr><td>INSERT</td><td>Inserts the rows into the table. It can be used only if the target table is empty.</td></tr><tr><td>APPEND</td><td>Appends new rows to the table if they do not conflict with existing tables.</td></tr><tr><td>REPLACE</td><td>Deletes existing rows in the table, using an SQL DELETE statement, before inserting the new rows.</td></tr></tbody></table>	INSERT	Inserts the rows into the table. It can be used only if the target table is empty.	APPEND	Appends new rows to the table if they do not conflict with existing tables.	REPLACE	Deletes existing rows in the table, using an SQL DELETE statement, before inserting the new rows.
INSERT	Inserts the rows into the table. It can be used only if the target table is empty.						
APPEND	Appends new rows to the table if they do not conflict with existing tables.						
REPLACE	Deletes existing rows in the table, using an SQL DELETE statement, before inserting the new rows.						

Property	Description
	TRUNCATE Deletes all existing rows in the table with an SQL TRUNCATE TABLE statement (Oracle Release 7.1 or later only).
Number of files to generate	The number of data, or combined control and data files to be generated. A value greater than 1 generates multiple distinct files that must be loaded using <i>sqlldr</i> with the DIRECT and PARALLEL keywords only. The default value is 1.
Size of row chunks in each data file	If the value of NUMFILES is greater than 1, this value determines how many rows are output to each of the distinct data files in turn. The default value is 1000.
Column name characters to map	A string of character pairs to map column names from their internal DataStage names to the column names used by the Oracle bulk loader. The format of this string is as follows: character 1 is mapped to character 2, character 3 is mapped to character 4, and so on. For example, the string <code>._</code> (period underscore), causes all periods in column names to be mapped to underscores. The string <code>aAbBcCdDeE</code> and so on maps lowercase characters to uppercase characters.

Any property value can contain a reference to a job parameter as *#param#*. This causes the current value of job parameter *param* to be substituted. This lets you specify different control and data file names for different runs of a DataStage job.

The BEFORE and AFTER subroutines can be used for a variety of purposes, such as notifying an operator that the files are available, or copying the files to the Oracle database and running *sqlldr*.

## Defining Character Set Maps

You can define a character set map for a particular Orabulk stage using the NLS page of the Stage dialog box. You can select a specific character set map from the list or accept the default setting for the whole project.

**Note:** The list contains all character set maps that are loaded and ready for use. You can view other maps that are supplied with DataStage by clicking **Show all maps**, but these maps cannot be used unless they are loaded using the DataStage Administrator. For more information, see *DataStage Administrator's Guide*.

## Loading the Files into Oracle

Once the control and data files have been completed, you can transfer them to the target Oracle database using the Oracle *sqlldr* command.

Before you run *sqlldr*, check that the following environment variables are set correctly:

ORACLE_HOME	This must be set to the Oracle installation directory.
ORACLE_SID	This must be set to the name of your Oracle database instance.
PATH	This must include the Oracle <i>bin</i> subdirectory.

## Running *sqlldr*

This section includes examples of syntax used for the *sqlldr* command running under UNIX. For more information see the Oracle manual *Oracle Server Utilities*. Every command line includes these syntax elements:

<i>username/password</i>	The Oracle user name and password, separated by a slash ( / ), to be used to log on to the database. <i>username</i> must have appropriate privileges for the database and the table to be loaded. Use UNIX shell quoting or escaping conventions to include white space or shell characters in the password.
<i>datafile</i>	Pathname of the <i>sqlldr</i> data file, or the combined control and data file.
<i>ctlfile</i>	Pathname of the <i>sqlldr</i> control file.
<i>logfile</i>	Pathname of a file for the <i>sqlldr</i> log.
<i>badfile</i>	Pathname into which <i>sqlldr</i> writes rows that could not be loaded. The rows are written in <i>sqlldr</i> format suitable for loading again when any problem has been resolved.

To load a single combined control and data file, use this command:

```
sqlldr userid=username/ password control=datafile log=logfile bad=badfile
```

To load separate single control and data files, use this command:

```
sqlldr userid=username/ password control=ctlfile data=datafile log=logfile  
bad=badfile
```

To load a single combined control and data file in Oracle DIRECT mode, add the DIRECT keyword as follows:

```
sqlldr userid=username/password control=datafile log=logfile bad=badfile  
direct=true
```

**Note:** DIRECT loading may require some administrator actions before and after loading to preserve database integrity. For more information, see *Oracle Server Utilities*.

To load multiple data files in parallel, run several *sqlldr* commands, each in its own UNIX process; for example, to load two data files in parallel, run the following two commands:

```
sqlldr userid=username/password control=ctlfile data=datafile1 log=logfile1  
bad=badfile1 direct=true parallel=true &
```

```
sqlldr userid=username/password control=ctlfile data=datafile2 log=logfile2  
bad=badfile2 direct=true parallel=true &
```

These could be generated with a shell script. Each command uses the same control file, but separate data files, log files, and bad files.

## Errors and Bad Rows

Any errors that occur during processing are reported in the DataStage log files. Possible errors include running out of disk space to write a control or data file, or insufficient operating system permissions to create and write these files. Errors that occur as the files are loaded into Oracle are recorded in the *sqlldr* log file.

Rejected rows are written to the bad file. The main reason for rejected rows is an integrity constraint in the target table; for example, null values in NOT NULL columns, nonunique values in UNIQUE columns, and so on. The bad file is in the same format as the input data file.

You can edit the file to correct errors and then resubmit it using the same control file. If the MODE property was not APPEND, you should edit the control file to change the REPLACE, TRUNCATE, or INSERT keyword to APPEND. This ensures that a subsequent *sqlldr* run does not delete rows you have already loaded. Each run of *sqlldr* appends new rejected rows to the bad file. You should manually truncate or delete the bad file after processing it so that rejected records from previous runs do not show up again.



## **Empty and Null Values**

Both empty and null values in input columns are loaded into Oracle as null values. If you examine the data file produced, you may find that some rows of values have fewer comma-delimited fields. These indicate trailing null values for columns not shown. If a target column is fixed-width, Oracle pads nonempty values on the right with blanks.



# 16

## Programming in DataStage

This chapter describes the programming tasks that you can perform in DataStage. At the heart of all programming in DataStage is the BASIC language. For more information about BASIC syntax, see “Syntax Help for Functions and Statements” in DataStage Developer’s Help.

There are several areas within DataStage where you may need to enter some code:

- Defining custom transforms. The function specified in a transform definition converts the data in a chosen column. See “Transforms” on page 11-15 for details.
- Defining custom routines to use as building blocks within other programming tasks. For example, you may define a routine which will then be reused by several custom transforms. This chapter tells you how to view, edit, and create your own BASIC routines using the DataStage Manager.
- Defining derivations, key expressions, and constraints while editing a Transformer stage. See “The DataStage Expression Editor” on page 11-12 for details.
- Defining before-stage and after-stage subroutines. These subroutines perform an action before or after a stage has processed data. These subroutines can be specified for Aggregator, Transformer, and some plug-in stages. See Chapter 11, Chapter 12, and Chapter 13 for details.
- Defining before-job and after-job subroutines. These subroutines perform an action before or after a job is run and are set as job properties. For more information, see “Job Properties” on page 5-30.

- **Defining job control routines.** These subroutines can be used to control other jobs from within the current job. For more information, see “Job Properties” on page 5-30.

## Programming Components

There are different types of programming components within DataStage. They fall within three broad categories:

- **Built-in.** DataStage comes with several built-in programming components that you can reuse within your jobs as required. Some of the built-in components are accessible using the DataStage Manager, and you can copy code from these. Others are only accessible from the Expression Editor, and the underlying code is not visible.
- **Custom.** You can also define your own programming components using the DataStage Manager, specifically routines and custom transforms. These are stored in the DataStage Repository and can be reused for other jobs and by other DataStage users.
- **External.** You can use certain types of external component from within DataStage. If you have a large investment in custom UniVerse functions or ActiveX (OLE) functions, then it is possible to call these from within DataStage. This is done by defining a wrapper routine which in turn calls the external functions. Note that the mechanism for including custom UniVerse functions is different from including ActiveX (OLE) functions. The former is described in this chapter, the latter under “Importing External ActiveX (OLE) Functions” on page 4-49.

The following sections discuss programming terms you will come across in DataStage.

## Routines

Routines are stored in the **Routines** branch of the DataStage Repository, where you can create, view or edit them using the Routine dialog box. The following program components are classified as routines:

- **Transform functions.** These are functions that you can use when defining custom transforms. DataStage has a number of built-in transform functions which are located in the **Routines > Examples > Functions** branch of the Repository. You can also define your own transform functions in the Routine dialog box.

- **Before/After subroutines.** When designing a job, you can specify a subroutine to run before or after the job, or before or after an active stage. DataStage has a number of built-in before/after subroutines, which are located in the **Routines ► Built-in ► Before/After** branch in the Repository. You can also define your own before/after subroutines using the Routine dialog box.
- **Custom UniVerse functions.** These are specialized BASIC functions that have been defined outside DataStage. Using the Routine dialog box, you can get DataStage to create a wrapper that enables you to call these functions from within DataStage. These functions are stored under the **Routines** branch in the Repository. You specify the category when you create the routine. If NLS is enabled, you should be aware of any mapping requirements when using custom UniVerse functions. If a function uses data in a particular character set, it is your responsibility to map the data to and from UNICODE.
- **ActiveX (OLE) functions.** You can use ActiveX (OLE) functions as programming components within DataStage. Such functions are made accessible to DataStage by importing them. This creates a wrapper that enables you to call the functions. After import, you can view and edit the BASIC wrapper using the Routine dialog box. By default, such functions are located in the **Routines ► Class name** branch in the Repository, but you can specify your own category when importing the functions.

When using the Expression Editor, all of these components appear under the **DS Routines...** command on the Suggest Operand menu.

A special case of routine is the job control routine. Such a routine is used to set up a DataStage job that controls other DataStage jobs. Job control routines are specified in the **Job control** page on the Job Properties dialog box. Job control routines are not stored under the **Routines** branch in the Repository.

## Functions

Functions take arguments and return a value. The word “function” is applied to many components in DataStage:

- **BASIC functions.** These are one of the fundamental building blocks of the BASIC language. When using the Expression Editor, you can access the BASIC functions via the **Function...** command on the Suggest Operand menu.
- **DataStage BASIC functions.** These are special BASIC functions that are specific to DataStage. These are mostly used in job control routines.

DataStage functions begin with DS to distinguish them from general BASIC functions. When using the Expression Editor, you can access the DataStage BASIC functions via the **DS Functions...** command on the Suggest Operand menu.

The following items, although called “functions,” are classified as routines and are described under “Routines” on page 16-2. When using the Expression Editor, they all appear under the **DS Routines...** command on the Suggest Operand menu.

- Transform functions
- Custom UniVerse functions
- ActiveX (OLE) functions

## Expressions

An expression is an element of code that defines a value. The word “expression” is used both as a specific part of BASIC syntax, and to describe portions of code that you can enter when defining a job. Areas of DataStage where you can use such expressions are:

- Defining breakpoints in the debugger
- Defining column derivations, key expressions and constraints in Transformer stages
- Defining a custom transform

In each of these cases the DataStage Expression Editor guides you as to what programming elements you can insert into the expression.

## Subroutines

A subroutine is a set of instructions that perform a specific task. Subroutines do not return a value. The word “subroutine” is used both as a specific part of BASIC syntax, but also to refer particularly to before/after subroutines which carry out tasks either before or after a job or an active stage. DataStage has many built-in before/after subroutines, or you can define your own.

Before/after subroutines are included under the general routine classification as they are accessible under the **Routines** branch in the Repository.

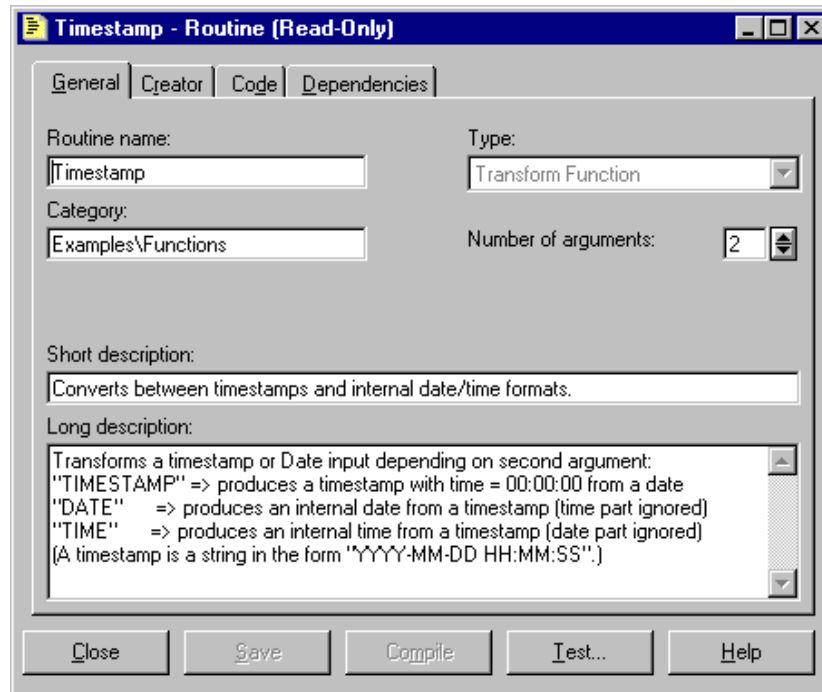
## Macros

DataStage has a number of built-in macros. These can be used in expressions, job control routines, and before/after subroutines. The available macros are concerned with ascertaining job status.

When using the Expression Editor, they all appear under the **DS Macro...** command on the Suggest Operand menu.

## The Routine Dialog Box

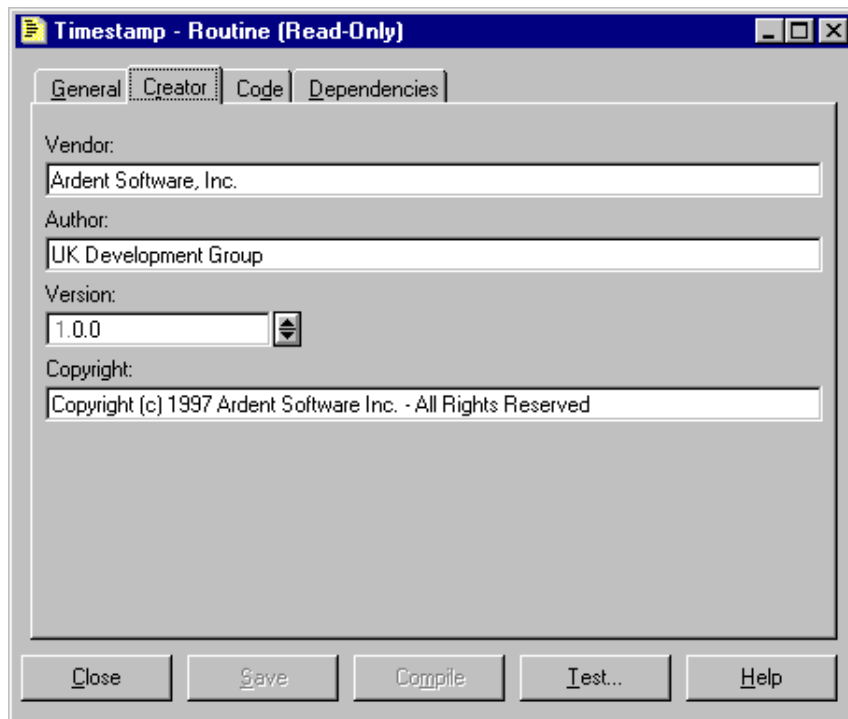
When you create, view, or edit a routine under the **Routines** branch in the DataStage Manager, the Routine dialog box appears:



This dialog box has four pages:

- **General.** Displayed by default. Contains general information about the routine, including:
  - **Routine name.** The name of the function or subroutine.

- **Type.** The type of routine. There are three types of routine: **Transform Function**, **Before/After Subroutine**, or **Custom UniVerse Function**.
- **Category.** The branch the routine is stored under in the Repository.
- **Number of arguments.** The maximum allowed is 10, the minimum is 1.
- **External Catalog Name.** This is only available if you have chosen **Custom UniVerse Function** from the Type box. Enter the cataloged name of the external routine.
- **Short description.** An optional brief description of the routine. The text entered in this field is displayed when you choose **View ► Details** from the DataStage Manager window or print a report.
- **Long description.** An optional detailed description of the routine.



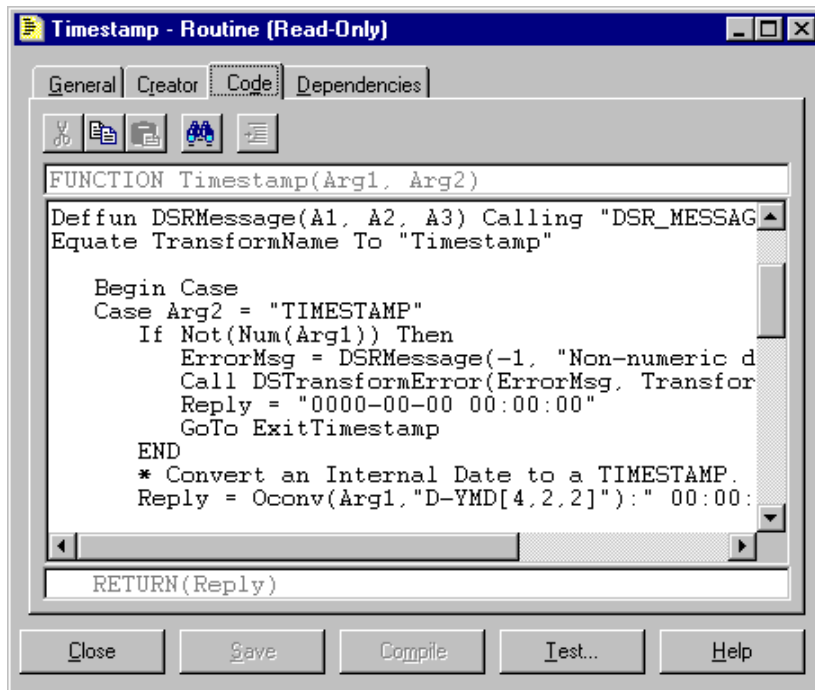
- **Creator.** Contains information about the creator and version number of the routine, including:
  - **Vendor.** The company who created the routine.



- **Author.** The creator of the routine.
- **Version.** The version number of the routine, which is used when the routine is imported. The **Version** field contains a three-part version number, for example, 3.1.1. The first part of this number is an internal number used to check compatibility between the routine and the DataStage system. The second part of this number represents the release number. This number should be incremented when major changes are made to the routine definition or the underlying code. The new release of the routine supersedes any previous release. Any jobs using the routine use the new release. The last part of this number marks intermediate releases when a minor change or fix has taken place.

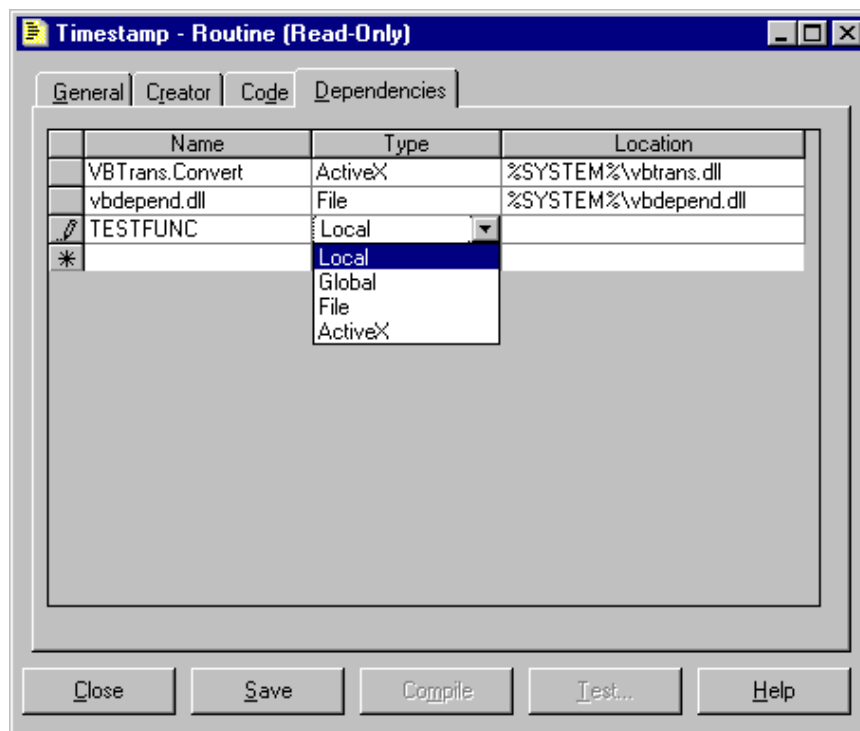
If you are creating a routine definition, the first part of the version number is set according to the version of DataStage you are using. You can edit the rest of the number to specify the release level. Click the part of the number you want to change and enter a number directly, or use the arrow button to increase the value.

- **Copyright.** Copyright information.



- **Code.** This page is used to view or write the code for the routine. The toolbar contains buttons for cutting, copying, pasting, and formatting code, and for activating Find (and Replace). The main part of this page consists of a multiline text box with scroll bars. For more information on how to use this page, see “Entering Code” on page 16-11.

**Note:** This page is not available if you selected **Custom UniVerse Function** on the **General** page.



- **Dependencies.** This page allows you to enter any locally or globally cataloged functions or routines that are used in the routine you are defining. This is to ensure that, when you package any jobs using this routine for deployment on another system, all the dependencies will be included in the package. The information required is as follows:
  - **Type.** The type of item upon which the routine depends. Choose from the following:
    - Local** Locally cataloged UniVerse BASIC functions and subroutines.

**Global** Globally cataloged UniVerse BASIC functions and subroutines.

**File** A standard file.

**ActiveX** An ActiveX (OLE) object (not available on UNIX-based systems).

- **Name.** The name of the function or routine. The name required varies according to the type of dependency:

**Local** The catalog name.

**Global** The catalog name.

**File** The file name.

**ActiveX** The **Name** entry is actually irrelevant for ActiveX objects. Enter something meaningful to you (ActiveX objects are identified by the **Location** field).

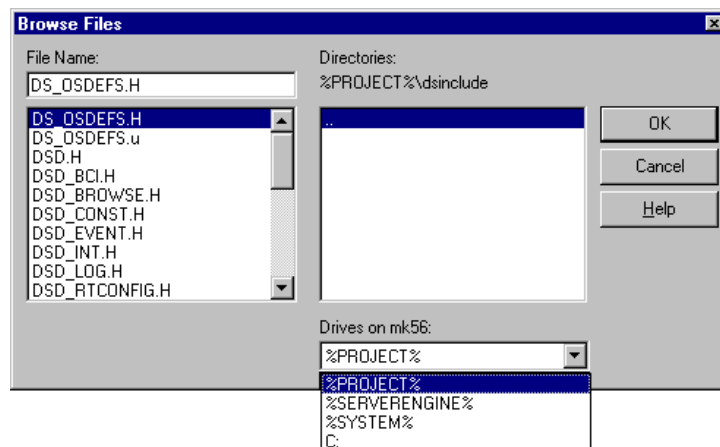
- **Location.** The location of the dependency. A browse dialog is available to help with this. This location can be an absolute path, but it is recommended you specify a relative path using the following environment variables:

**%SERVERENGINE%** – UniVerse UV account directory (normally *C:\Ardent\DataStage\ServerEngine*).

**%PROJECT%** – Current project directory.

**%SYSTEM%** – System directory on Windows NT or */usr/lib* on UNIX.

The Browse Files dialog box is shown below. You cannot navigate to the parent directory of an environment variable.



When browsing for the location of a file on a UNIX server, there is an entry called **Root** in the **Base Locations** drop-down list.

There are five buttons in the Routine dialog box. Their availability depends on the action you are performing and the type of routine you are editing.

- **Close.** Closes the Routine dialog box. If you have any unsaved changes, you are prompted to save them.
- **Save.** Saves the routine.
- **Compile...** . Compiles a saved routine. This button is available only when there are no outstanding (unsaved) changes.
- **Test...** . Tests a routine. This button is available only for routines of type **Transform Function** and **Custom UniVerse Function**. This is because you cannot test before-subroutines and after-subroutines in isolation. This button is active only when the routine has compiled or referenced successfully.
- **Help.** Invokes the Help system.

## Creating a Routine

To create a new routine, select the **Routines** branch in the DataStage Manager window and do one of the following:

- Choose **File ► New Routine...** .
- Choose **New Routine...** from the shortcut menu.
- Click the **New** icon on the toolbar.

The Routine dialog box appears. On the **General** page:

1. Enter the name of the function or subroutine in the **Routine name** field. This should not be the same as any BASIC function name.
2. Choose the type of routine you want to create from the Type drop-down list box. There are three options:
  - **Transform Function.** Choose this if you want to create a routine for a Transform definition.
  - **Before/After Subroutine.** Choose this if you want to create a routine for a before-stage or after-stage subroutine or a before-job or after-job subroutine.

- **Custom UniVerse Function.** Choose this if you want to refer to an external routine, rather than define one in this dialog. If you choose this, the **Code** page will not be available.
3. Enter a category name in the **Category** field. This name is used to create a branch under the main **Routines** branch. If you do not enter a name in this field, the routine is created under the main **Routines** branch.
  4. Specify the number of arguments in the **Number of arguments** box. The number of arguments is set to 2 (and cannot be edited) for a routine type of **Before/After Subroutine**. The number of arguments is set to 1 by default for a routine type of **Transform Function** or **External Function**, but you can increase this number using the arrow buttons.
  5. Optionally enter a brief description of the routine in the **Short description** field. The text entered here is displayed when you choose **View ► Details** from the DataStage Manager window.
  6. Optionally enter a more detailed description of the routine in the **Long description** field.

Once this page is complete, you can enter creator information on the **Creator** page, but this is optional. You must then enter your code on the **Code** page.

## Entering Code

You can enter or edit code for a routine on the **Code** page in the Routine dialog box.

The first field on this page displays the routine name and the number of arguments. If you want to change these properties, you must edit the fields on the **General** page.

The main part of this page contains a multiline text entry box, in which you must enter your code. To enter code, click in the box and start typing. You can use the following standard Windows edit functions in this text box:

- Delete using the **Del** key
- Cut using **Ctrl-X**
- Copy using **Ctrl-C**
- Paste using **Ctrl-V**
- Go to the end of the line using the **End** key
- Go to the beginning of the line using the **Home** key
- Select text by clicking and dragging or double-clicking

Some of these edit functions are included in a shortcut menu which you can display by clicking the right mouse button. You can also cut, copy, and paste code using the icons in the toolbar.

Your code must only contain BASIC functions and statements supported by DataStage. If you are unsure of the supported functions and statements, or the correct syntax to use, see DataStage Developer's Help for a complete list of supported DataStage BASIC functions.

If NLS is enabled, you can use non-English characters in the following circumstances:

- In comments
- In string data (that is, strings contained in quotation marks)

The use of non-English characters elsewhere causes compilation errors.

If you want to format your code, click the **Format** icon on the toolbar.

The last field on this page displays the return statement for the function or subroutine. You cannot edit this field.

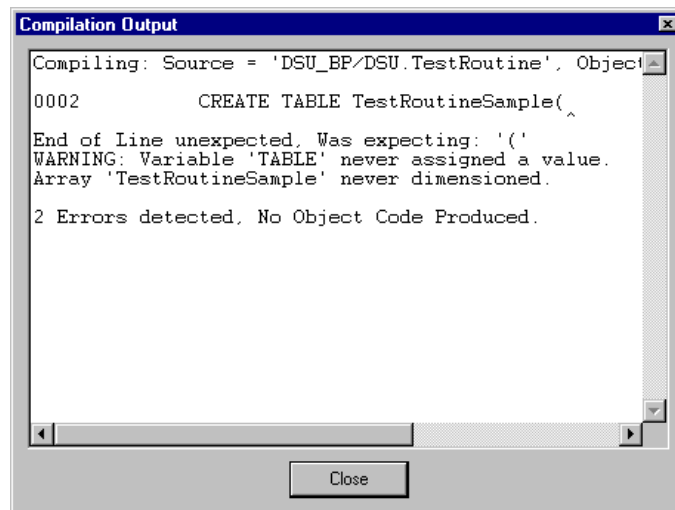
## Saving Code

When you have finished entering or editing your code, the routine must be saved. A routine cannot be compiled or tested if it has not been saved. To save a routine, click **Save** in the Routine dialog box. The routine properties (its name, description, number of arguments, and creator information) and the associated code are saved in the Repository.

## Compiling Code

When you have saved your routine, you must compile it. To compile a routine, click **Compile...** in the Routine dialog box. If the routine compiles successfully, a message box appears. Click **OK** to acknowledge the message. The routine is marked as "built" in the Repository and is available for use. If the routine is a **Transform Function**, it is displayed in the list of available functions when you edit a transform. If the routine is a **Before/After Subroutine**, it is displayed in the drop-down list box of available subroutines when you edit an Aggregator, Transformer, or plug-in stage, or define job properties.

If the routine failed to compile, the Compilation Output window appears displaying the errors generated:



Before you start to investigate the source of the error, you may find it useful to move the Compilation Output window alongside or below the Routine dialog box, as you need to see both windows to troubleshoot the error.

To troubleshoot the error, double-click the error in the Compilation Output window. DataStage attempts to find the corresponding line of code that caused the error and highlights it in the Routine dialog box. You must edit the code to remove any incorrect statements or to correct any syntax errors.

If NLS is enabled, watch for multiple question marks in the Compilation Output window. This generally indicates that a character set mapping error has occurred.

When you have modified your code, click **Save** then **Compile...** . If necessary, continue to troubleshoot any errors, until the routine compiles successfully.

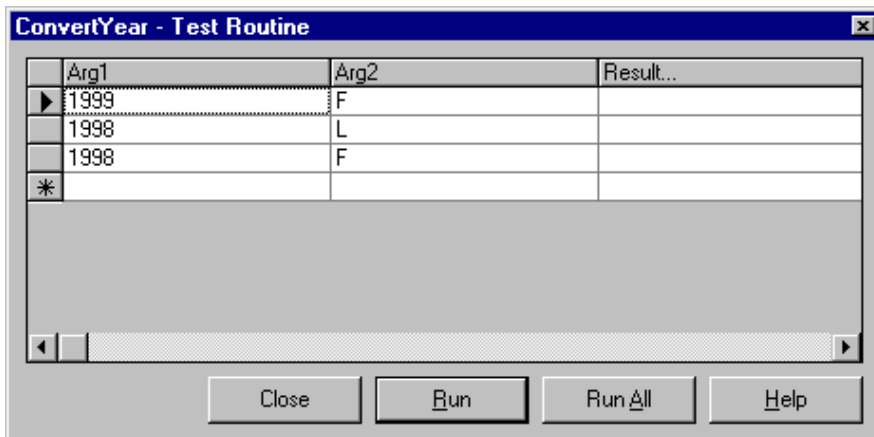
Once the routine is compiled, you can use it in other areas of DataStage or test it. For more information, see “Testing a Routine” on page 16-13.

## Testing a Routine

Before using a compiled routine, you can test it using the **Test...** button in the Routine dialog box. The **Test...** button is activated when the routine has been successfully compiled.

**Note:** The **Test...** button is not available for a **Before/After Subroutine**. Routines of this type cannot be tested in isolation and must be executed as part of a running job.

When you click **Test...**, the Test Routine dialog box appears:



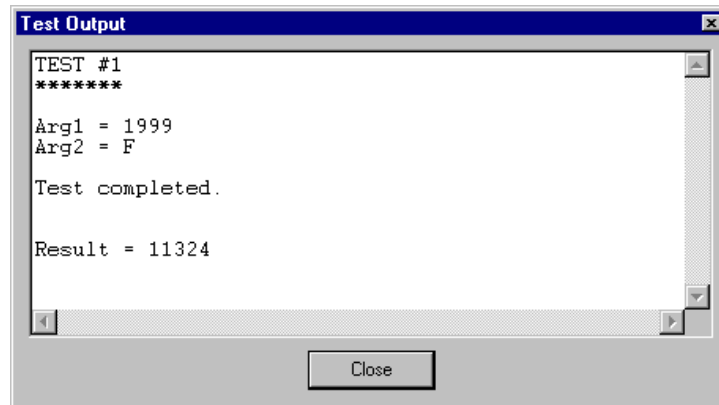
This dialog box contains a grid and buttons. The grid has a column for each argument (called Arg1, Arg2, ... Arg $n$ ) and one for the test result.

You can add and edit rows in the grid to specify the values for different test cases. For more information about using and editing a grid, see Appendix B, “Editing Grids.”

To run a test with a chosen set of values, click anywhere in the row you want to use and click **Run**. If you want to run tests using all the test values, click **Run All**. The **Result...** column is populated as each test is completed.



To see more details for a particular test, double-click the **Result...** cell for the test you are interested in. The Test Output window appears, displaying the full test results:



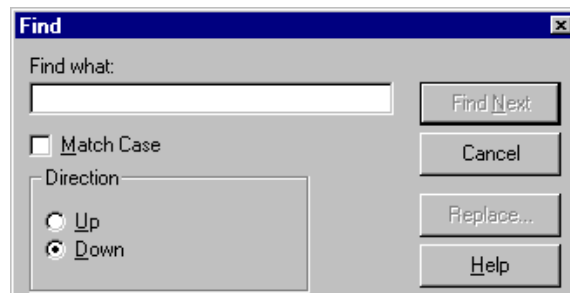
Click **Close** to close this window.

If you want to delete a set of test values, click anywhere in the row you want to remove and press the **Delete** key or choose **Delete row** from the shortcut menu.

When you have finished testing the routine, click **Close** to close the Test Routine dialog box. Any test values you entered are saved when you close the dialog box.

## Using Find and Replace

If you want to search the code for specific text, or replace text, you can use **Find** and **Replace**. To start **Find**, click the **Find** icon on the **Code** page toolbar. The Find dialog box appears:

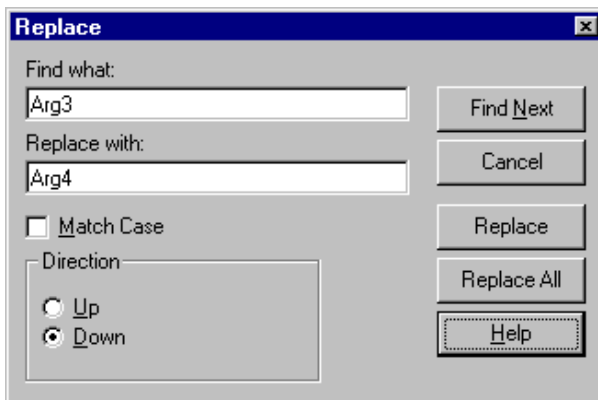


This dialog box has the following fields, options, and buttons:

- **Find what.** Contains the text to search for. Enter appropriate text in this field. If text was highlighted in the code before you chose **Find**, this field displays the highlighted text.
- **Match case.** Specifies whether to do a case-sensitive search. By default this check box is cleared. Select this check box to do a case-sensitive search.
- **Up and Down.** Specifies the direction of search. The default setting is **Down**. Click **Up** to search in the opposite direction.
- **Find Next.** Starts the search. This button is unavailable until you specify text to search for. Continue to click **Find Next** until all occurrences of the text have been found.
- **Cancel.** Closes the Find dialog box.
- **Replace...** . Displays the Replace dialog box. For more information, see “Replacing Text” on page 16-16.
- **Help.** Invokes the Help system.

## Replacing Text

If you want to replace text in your code with an alternative text string, click the **Replace...** button in the Find dialog box. When you click this button, the Find dialog box changes to the Replace dialog box:



This dialog box has the following fields, options, and buttons:

- **Find what.** Contains the text to search for and replace.

- **Replace with.** Contains the text you want to use in place of the search text.
- **Match case.** Specifies whether to do a case-sensitive search. By default this check box is cleared. Select this check box to do a case-sensitive search.
- **Up and Down.** Specifies the direction of search and replace. The default setting is **Down**. Click **Up** to search in the opposite direction.
- **Find Next.** Starts the search and replace. This button is unavailable until you specify text to search for. Continue to click **Find Next** until all occurrences of the text have been found.
- **Cancel.** Closes the Replace dialog box.
- **Replace.** Replaces the search text with the alternative text.
- **Replace All.** Performs a global replace of all instances of the search text.
- **Help.** Invokes the Help system.

## Viewing and Editing a Routine

You can view and edit any user-written functions and subroutines in your project. To view or modify a function or subroutine, select the function or subroutine in the display area and do one of the following:

- Choose **File ► Properties...**
- Choose **Properties...** from the shortcut menu.
- Click the **Properties** icon on the toolbar.
- Double-click the function or subroutine in the display area.

The Routine dialog box appears. You can edit any of the fields and options on any of the pages. If you make any changes, you must save, compile, and test the code before closing the Routine dialog box. See earlier in this chapter for more information.

## Copying a Routine

You can copy an existing routine using the DataStage Manager. To copy a routine, select it in the display area and do one of the following:

- Choose **File ► Copy**.
- Choose **Copy** from the shortcut menu.
- Click the **Copy** icon on the toolbar.

The routine is copied and a new routine is created under the same branch in the project tree. By default, the name of the copy is called CopyOfXXX, where XXX is the name of the chosen routine. An edit box appears allowing you to rename the copy immediately. The new routine must be compiled before it can be used.

## Renaming a Routine

You can rename any of the existing routines using the DataStage Manager. To rename an item, select it in the display area and do one of the following:

- Click the routine again. An edit box appears and you can enter a different name or edit the existing one. Save the new name by pressing **Enter** or by clicking outside the edit box.
- Choose **File ► Rename**. An edit box appears and you can enter a different name or edit the existing one. Save the new name by pressing **Enter** or by clicking outside the edit box.
- Choose **Rename** from the shortcut menu. An edit box appears and you can enter a different name or edit the existing one. Save the new name by pressing **Enter** or by clicking outside the edit box.
- Double-click the routine. The Routine dialog box appears and you can edit the **Routine name** field. Click **Save**, then **Close**.

# 17

## Debugging, Compiling, and Releasing a Job

This chapter describes how to create and release an executable job. When you have edited all the stages in a job design, you can create an executable job by compiling your job design. The debugger helps you to iron out any problems in your design. The job can then be validated and run using the DataStage Director.

If you want to ship the executable job to another DataStage system, you must release the job before packaging it for deployment. For more information about packaging a job, see Chapter 19, “Importing, Exporting, and Packaging Jobs.”

### The DataStage Debugger

The DataStage debugger provides basic facilities for testing and debugging your job designs. The debugger is run from the DataStage Designer. It can be used from a number of places within the Designer:

- Debug menu (**File** ► **Debug**)
- Debugger toolbar
- Shortcut menu (some items)

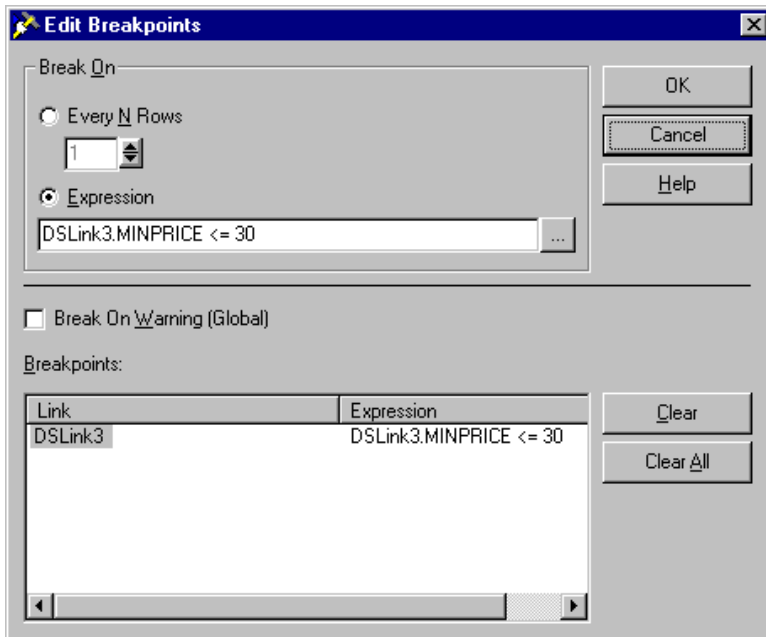
The debugger enables you to set breakpoints on the links in your job. When you run the job in debug mode, the job will stop when it reaches a breakpoint. You can then step to the next action (reading or writing) on that link, or step to the processing of the next row of data (which may be on the same link or another link).

Any breakpoints you have set remain if the job is closed and reopened. Breakpoints are validated when the job is compiled, and remain valid if the link to which it belongs is moved, or has either end moved, or is renamed. If, however, a link is deleted and another of the same name created, the new link does not inherit the breakpoint. Breakpoints are not inherited when a job is saved under a different name, exported, or upgraded.

To add a breakpoint:

1. Select the required link.
2. Select **Toggle Breakpoint** from the Debugger menu or the Debugger toolbar. The breakpoint can subsequently be removed by selecting **Toggle Breakpoint** again.

A circle appears on the link to indicate that a breakpoint has been added.

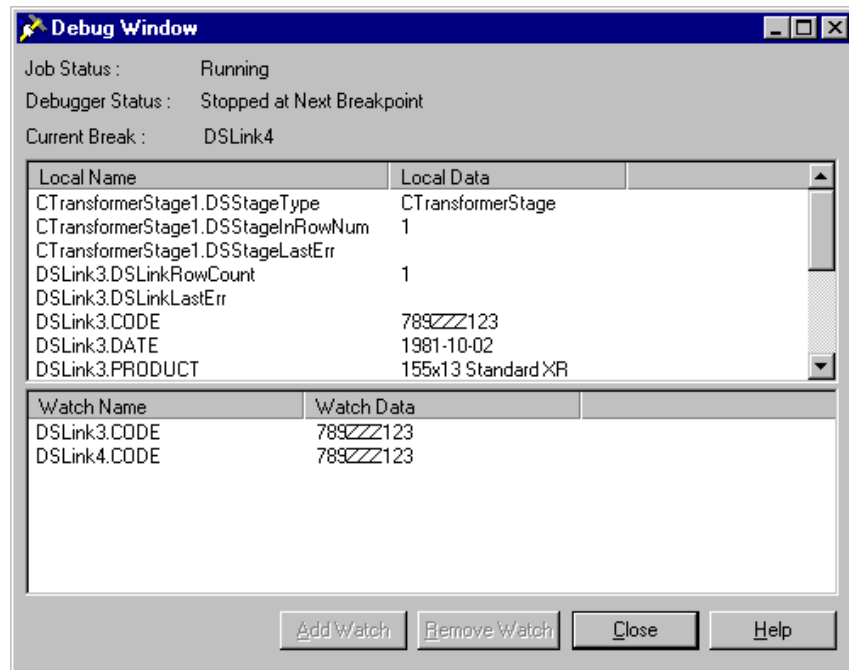


**Note:** You cannot place a breakpoint on a link which has a container as its source stage. Instead, you should place the breakpoint on that same link as represented in the container view. The link will only be shown as having a breakpoint in the container view.

The Debug Window allows you to view variables in the watch list and any in-context variables when you stop at a breakpoint.

The Debug Window is visible whenever **View ► Debug Window** or **File ► Debug ► Debug Window** is selected. It always appears on the top of the DataStage Designer window. Right-clicking in the Debug Window invokes a shortcut menu containing the same items as the Debug menu. The Debug Window has two

display panes. You can drag the splitter bar between the two panes to resize them relative to one another. The window also gives information about the status of the job and debugger.



The upper pane shows local variables. Before debugging starts, all the columns on all the links in the job are displayed, and all are marked “Out of context”. During debugging, the pane shows only the variables that are in context when the job is stopped at a breakpoint. It displays the names and values of any variables currently in context and you can add any of these variables to the watch list, which maintains a record of selected variables for as long as required.

The lower pane displays variables in the watch list. When variables are in context, their values are displayed and updated at every breakpoint. When variables are out of context, they are marked “Out of context”. The watch list is saved between sessions.

To add a variable to the watch list:

1. Select the variable name in the upper pane of the Debug Window.
2. Click **Add Watch**. The variable will be added to the watch list and will appear in the lower pane.

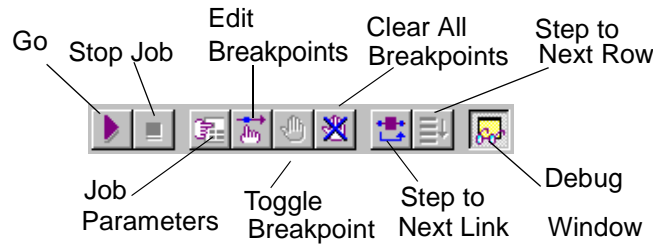
To delete variables from the watch list, select the variables and click **Remove Watch**.

The following commands are available from the Debug menu or Debug toolbar:

- **Go.** Runs the current job in debug mode, compiling it first if necessary. In debug mode the job will run until a breakpoint is encountered. It then stops in break mode, allowing you to interact with the job. The first time that **Go** is used after a job is compiled or loaded, the Job Run Options dialog box appears and collects any required parameter values or run-time limits (see “The Job Run Options Dialog Box” on page 5-26).
- **Stop Job.** Only available in break mode. Stops the job and exits break mode.
- **Job Parameters...** . Allows you to specify job parameters for when the job is run in debug mode. Selecting this invokes the Job Run Options dialog box, allowing you to specify any required parameters or run-time limits for the job. See “The Job Run Options Dialog Box” on page 5-26. The item is disabled once the job is started in debug mode.
- **Edit Breakpoints...** . Allows you to edit existing breakpoints or add new ones.
- **Toggle Breakpoint.** Allows you to set or clear a breakpoint from the selected link. If a link has a breakpoint set (indicated by a dark circle at the link source), then **Toggle Breakpoint** clears that breakpoint. If the link has no breakpoint, then one is added, specifying a stop at every row processed.
- **Clear All Breakpoints.** Deletes all breakpoints defined for all links.
- **Step to Next Link.** This causes the job to run until the next action occurs on any link (reading or writing), when it stops in break mode.
- **Step to Next Row.** This causes the job to run until the next row is processed or until another link with a breakpoint is encountered, whichever comes first. The job then stops in break mode. If the job is not currently stopped at a breakpoint on a link (for example, if it hasn’t started debugging yet, or is stopped at a warning), then this will perform as **Step to Next Link**.



- **Debug Window.** Select this to display the Debug Window. Deselect it to hide the Debug Window.



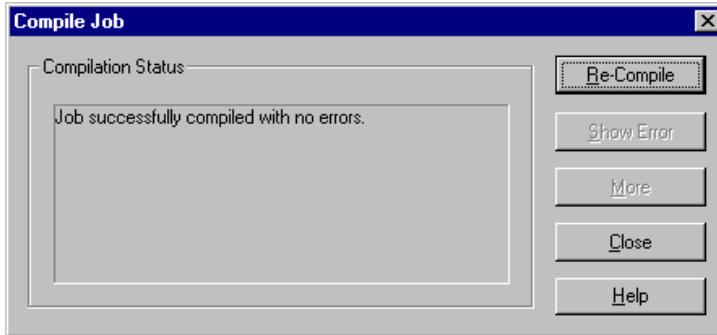
## Compiling a Job

Jobs are compiled using the DataStage Designer. To compile a job, open the job in the Designer and do one of the following:

- Choose **File ► Compile**.
- Click the **Compile** icon on the toolbar.

If the job has unsaved changes, you are prompted to save the job by clicking **OK**. The Compile Job window appears. This window contains a display area for compilation messages and has the following buttons:

- **Re-Compile.** Recompiles the job if you have made any changes.
- **Show Error.** Highlights the stage that generated a compilation error. This button is only active if an error is generated during compilation.
- **More.** Displays the output that does not fit in the display area. Some errors produced by the compiler include detailed BASIC output.
- **Close.** Closes the Compile Job window.
- **Help.** Invokes the Help system.



The job is compiled as soon as this window appears. You must check the display area for any compilation messages or errors that are generated.

If there have been breakpoints set for links that no longer exist, a message appears during compilation warning you of this. The breakpoints are then automatically removed.

## Compilation Checks

During compilation, the following criteria in the job design are checked:

- **Primary Input.** If you have more than one input link to a Transformer stage, the compiler checks that one is defined as the primary input link.
- **Reference Input.** If you have reference inputs defined in a Transformer stage, the compiler checks that these are not from sequential files.
- **Key Expressions.** If you have key fields specified in your column definitions, the compiler checks that there are key expressions joining the data tables.
- **Transforms.** If you have specified a transform, the compiler checks that this is a suitable transform for the data type.

## Successful Compilation

If the Compile Job window displays the message `Job successfully compiled with no errors.` you can:

- Validate the job
- Run or schedule the job
- Release the job
- Package the job for deployment on other DataStage systems

Jobs are validated and run using the DataStage Director. See *DataStage Operator's Guide* for more information.

## Troubleshooting

If the Compile Job window displays an error, you can use the **Show Error** button to troubleshoot your job design. When you click the **Show Error** button, the stage that contains the first error in the design is highlighted. You must edit the stage to change any incorrect settings and recompile.

The process of troubleshooting compilation errors is an iterative process. You must refine each “problem” stage until the job compiles successfully.

## Releasing a Job

If you are developing a job for users on another DataStage system, you must label the job as ready for deployment before you can package it. For more information about packaging a job, see Chapter 19, “Importing, Exporting, and Packaging Jobs.”

To label a job for deployment, you must release it. A job can be released when it has been compiled and validated successfully at least once in its life.

Jobs are released using the DataStage Manager. To release a job:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box.
2. Double-click the **Jobs** branch in the project tree.
3. Select the job you want to release in the display area.
4. Choose **Tools ► Release Job**. The Job Release dialog box appears, which shows a tree-type hierarchy of the job and any associated dependent jobs.
5. Select the job that you want to release.
6. Click **Release Job** to release the selected job, or **Release All** to release all the jobs in the tree.

A physical copy of the chosen job is made (along with all the routines and code required to run the job) and it is recompiled. The Releasing Job dialog box appears and shows the progress of the releasing process.

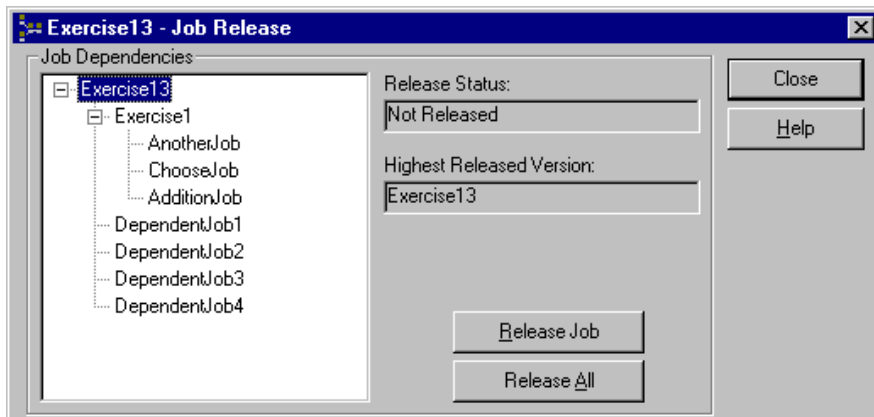
The released job is automatically assigned a name and version number using the format *jobname%reln.n.n*. *jobname* is the name of the job you chose to release and *n.n.n* is the version number. When you refer to a job by its released name, this is known as a “fixed job release,” which always equates to that particular version of the job.

You can use the Designer to view the design of a released job. However, if you edit the job design you cannot save the changes. The meta data and settings displayed in the job design are stored as part of the released job and these may not match the information currently held in the Repository. This is especially true if you developed the table definitions, transforms, routines, or job design after the job was released.

If you want to develop and enhance a job design, you must edit the original job. To use the changes you have made, you must release the job again.

**Note:** Released jobs cannot be copied or renamed using the Manager.

The Job Release dialog box is shown below:



This dialog box contains a tree-type hierarchy showing the job dependencies of the job you are releasing. It displays the status of the selected job as follows:

- **Not Compiled.** The job exists, but has not yet been compiled (this means you will not be able to release it).
- **Not Released.** The job has been compiled, but not yet released.
- **Job Not Found.** The job cannot be found.

- **Released.** The job has previously been released.
- **Release Exists.** The selected job is a fixed version (i.e., has a particular release number) and that version of the job exists.

The dialog box also displays the highest released version of the selected job. When the selected job is a fixed version job (i.e., has a particular release number), then it displays **Fixed Job Release**.

If a dependent job appears in more than one branch of the hierarchy, then only the one at the highest level is displayed.



# 18

## Reporting and Printing

This chapter describes how to generate reports from the DataStage Manager. It also describes how to print from the Designer.

### Reporting

The DataStage Reporting Tool is flexible and allows you to generate reports at various levels within a project, for example, entire job, single stage, set of stages, etc.

Information generated for reporting purposes is stored in a relational database on the DataStage client. This information can then be used to print a report, write a report to a file, or be interrogated by a third-party tool.

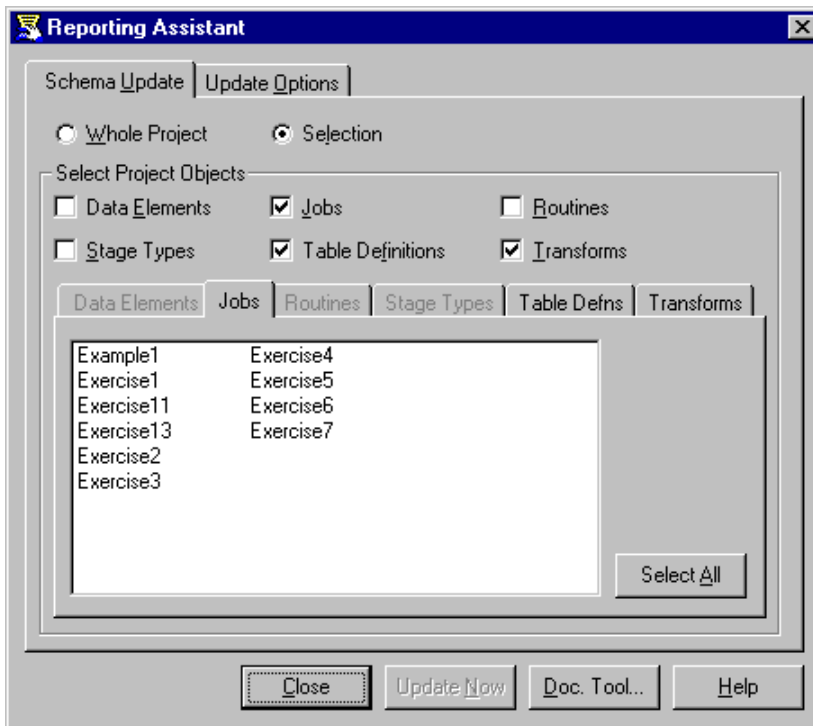
A Microsoft Access database is provided on the DataStage client for this purpose. It offers a number of predefined report formats for you to choose from. You can extend or customize this database, or use some other SQL database for these purposes. It is possible to target any SQL database which is supported by a Level 2-compliant ODBC driver on the client machine. Create database scripts for most popular databases are provided in the main client directory on the DataStage client (by default, `C:\ardent\datastage\clients`).

The database used for reporting purposes must use the same language as the DataStage clients.

### The Reporting Tool

The DataStage Reporting Tool is invoked by choosing **Tools ► Reporting Assistant...** from the DataStage Manager. The Reporting Assistant dialog box appears. This dialog box has two pages: **Schema Update** and **Update Options**.

**Note:** If you want to use the Reporting Tool you should ensure that the names of your DataStage components (jobs, stages, links, etc.) do not exceed 64 characters.

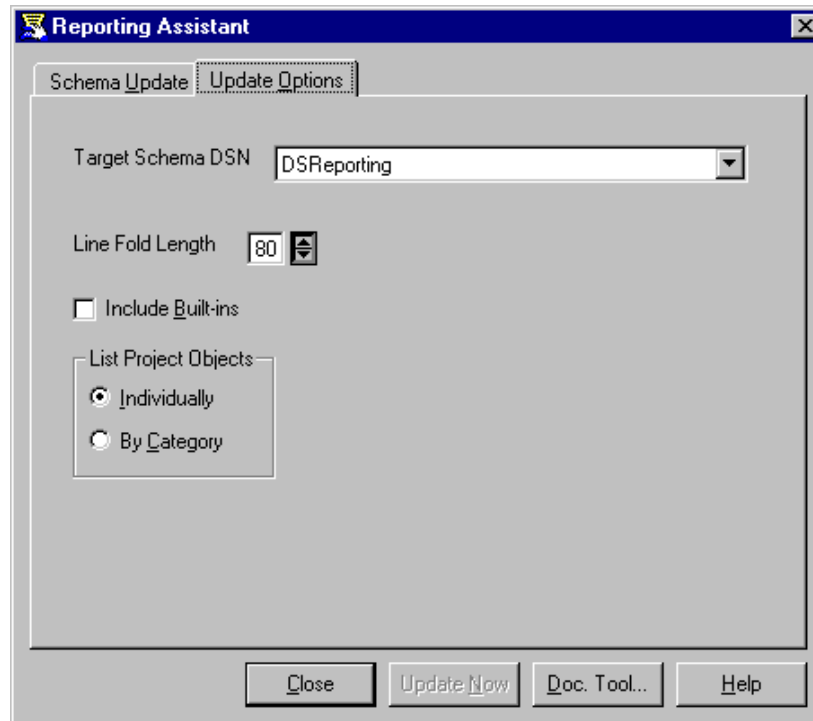


The **Schema Update** page allows you to specify what details in the reporting database should be updated and when. This page contains the following fields and buttons:

- **Whole Project.** Click this button if you want all project details to be updated in the reporting database. If you select **Whole Project**, other fields in the dialog box are disabled. This is selected by default.
- **Selection.** Click this button to specify that you want to update selected objects. The Select Project Objects area is then enabled.
- **Select Project Objects.** Select the check boxes of all objects that you want to be updated in the reporting database. The corresponding sub-tab is then enabled. For example, if you select the **Jobs** check box, you can go on to specify which jobs to report on in the **Jobs** page.



The **Update Options** page allows you to make adjustments to any requested updates to the reporting database.



This page contains the following fields and buttons:

- **Target Schema DSN.** Select the required database from the drop-down list box. All current DSNs available on the client machine are listed. The MS Access database supplied with DataStage is selected by default.
- **Line Fold Length.** Select a value to determine where carriage returns will be inserted in potentially long properties. For example, the Description property may well run to hundreds of characters, so selecting 80 would specify that carriage returns are inserted every 80 characters. If there are already carriage returns in the text, these will be used in preference.
- **Include Built-ins.** Specifies that built-in objects should be included in the update.

- **List Project Objects.** Determines how objects are displayed in the lists on the **Schema Update** page:
  - **Individually.** This is the default. Lists individual objects of that type, allowing you to choose one or more objects. For example, every data element would be listed by name: **Number**, **String**, **Time**, **DATA.TAG**, **MONTH.TAG**, etc.
  - **By Category.** Lists objects by category, allowing you to choose a particular category of objects for which to update the details. For example, in the case of data elements, you might choose among **All**, **Built-in**, **Built-in/Base**, **Built-in/Dates**, and **Conversion**.

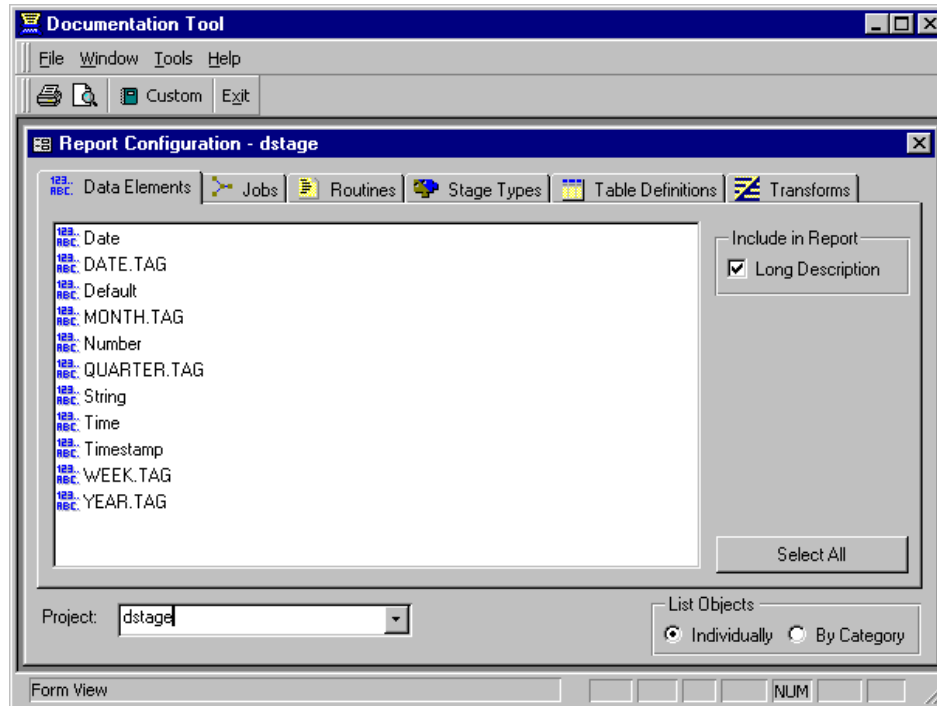
The following buttons are on both pages of the Reporting Assistant dialog box:

- **Update Now.** Updates the selected details in the reporting database. Note that this button is disabled if no details are selected in the **Schema Update** page.
- **Doc. Tool...** . Invokes the Documentation Tool dialog box in order to request a report.

## The Documentation Tool

The Documentation Tool dialog box is invoked from the **Doc. Tool...** button on the Reporting Assistant dialog box. It allows you to print predefined reports from the Microsoft Access database. The dialog box remains in view until you click **Exit**.

Each page has an Include in Report area which allows you to choose options for various types of report. Click the **Preview** icon in the toolbar to see what a finished report will look like.



The toolbar has the following functions:

- **Print.** Prints a report of the format specified in the Report Configuration form.
- **Print Preview.** Gets a preview of the form before you actually print it. To print from the preview, click the **Print** button.
- **Custom Reports.** Gets a list of available reports. Use this to access reports added to the MS Access Documentation Tool besides those supplied with DataStage.
- **Exit.** Exits the Documentation Tool when you have printed all required reports.

The Documentation Tool dialog box has a Report Configuration form which in turn has a page for every type of object. Each page has a list box that lists all objects of that type currently in the reporting database. Objects can be listed individually or by category (note that jobs can only be listed individually).

The Report Configuration form has the following fields and buttons:

- **Include in Report.** This area lists the optional fields that can be excluded or included in the report. Check the box to include an item in the report, clear it to exclude it. All items are checked by default.
- **Select All.** Selects all items in the list window.
- **Project.** Lists all the available projects. Choose the project that you want to report on.
- **List Objects.** This area determines how objects are displayed in the lists on the **Schema Update** page:
  - **Individually.** This is the default. Lists individual objects of that type, allowing you to choose one or more objects. For example, every data element would be listed by name: **Number, String, Time, DATA.TAG, MONTH.TAG**, etc.
  - **By Category.** Lists objects by category, allowing you to choose a particular category of objects for which to update the details. For example, in the case of data elements, you might choose among **All, Built-in, Built-in/Base, Built-in/Dates**, and **Conversion**.

Examples of some of the reports are given below:

---

## Data Elements Listing

from project acc1 on server mk74

---

**Data Element Name** DataElement1

**Category** Cat1

**Short Description** shorty

**Long Description**

long desc

<b>Base Type</b>	<b>SQL Type</b>	<b>Precision</b>	<b>Scale</b>	<b>Nullable</b>	<b>Display Size</b>
Default	Char	10		Yes	10

---

**Date Modified** 02-Feb-98                      **Last Updated** 06-Feb-98

---

**Data Element Name** DataElement2

**Category** Cat1/Cat2

**Short Description** sh

**Long Description**

long desc

<b>Base Type</b>	<b>SQL Type</b>	<b>Precision</b>	<b>Scale</b>	<b>Nullable</b>	<b>Display Size</b>
Number	Integer	4	2	No	4

---

**Date Modified** 02-Feb-98                      **Last Updated** 06-Feb-98

---

**Data Element Name** DataElement3

**Category** Cat1/Cat2/Cat3

**Short Description** short desc

**Long Description**

long desc

<b>Base Type</b>	<b>SQL Type</b>	<b>Precision</b>	<b>Scale</b>	<b>Nullable</b>	<b>Display Size</b>
String	VarChar	33000		Yes	255

---

**Date Modified** 02-Feb-98                      **Last Updated** 06-Feb-98

---

**End of Data Elements Listing**

---

---

## Routines Listing

from project acc1 on server mk74

---

**Routine Name** TestRoutine  
**Category** cat1/cat2/cat3  
**Type** Transform Function **No. of Arguments** 2  
**Short Description** a short desc  
**Long Description**  
This is a long  
description  
**Code**  
This is some code  
**Vendor** vendor  
**Author** author  
**Version** 6.1.4  
**Copyright** copyright

---

**Date Modified** 23-Jan-98 **Last Updated** 05-Feb-98

---

**Routine Name** TestRoutine2  
**Category** Examples/Functions  
**Type** Before/After Subroutine **No. of Arguments** 0  
**Short Description**  
**Long Description**  
This is a  
long description  
**Code**  
This is some code  
**Vendor** Ardent Software, Inc.  
**Author** UK Development Group  
**Version** 0.0.0  
**Copyright** Copyright (c) 1997 Ardent Software Inc. - All Rights Reserved

---

**Date Modified** 27-Jan-98 **Last Updated** 05-Feb-98

---

**End of Routines Listing**

---

---

## Transforms Listing

from project acc1 on server mk74

---

**Transform Name** Transform1  
**Category** Martins  
**Short Description** short desc  
**Long Description**  
this is a long description  
**Source Data Element** DESource  
**Target Data Element** DETarget  
**Function Name** Funky1  
**Is Function 'User Written'?** Yes      **No. of Arguments in Function** 3  
**Additional Argument List** 1, "silly", true

---

**Date Modified** 02-Feb-98      **Last Updated** 06-Feb-98

---

**Transform Name** Transform2  
**Category** Martins/Experiment  
**Short Description** short desc  
**Long Description**  
long desc  
**Source Data Element** DESource  
**Target Data Element** DETarget  
**Function Name** Funky2  
**Is Function 'User Written'?** No      **No. of Arguments in Function** 1  
**Additional Argument List**

---

**Date Modified** 24-Jan-98      **Last Updated** 01-Feb-98

---

**End of Transforms Listing**

---

## Printing from the Designer

You can print the contents of the current Diagram window from the Designer. The printing format and the printer used are determined by:

- The printer setup. The printer and paper source are determined by your default Windows printer.
- The default print options. The print orientation is determined by the settings in the Designer Options dialog box.

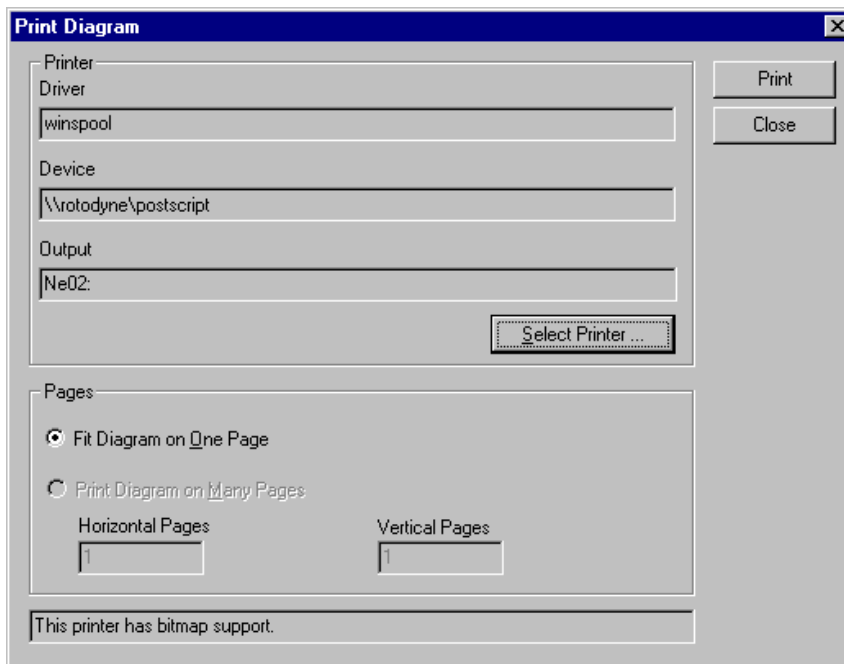
The print lines display the area to be printed. If necessary, use the **Zoom out** icon on the toolbar to reduce the display until you see the print lines.

## Producing a Printout

To print the contents of the Diagram window, do one of the following:

- Choose **File** ► **Print...**
- Click the **Print** icon on the toolbar.

The Print Diagram dialog box appears:



You can change the print settings using the **Select Printer...** button. To accept the printer settings and to produce the printout, click **Print**.

## Changing the Printer Setup

You can change the printer and the paper supply by clicking the **Select Printer...** button in the Print Diagram dialog box. The Print Setup dialog box appears. This dialog box contains all the settings for your default Windows printer. You can change any of the settings.

Click **OK** to save any changes and to close the Print Setup dialog box.



**Note:** The print orientation is set by the Designer options. If you change the print orientation in the Print Setup dialog box, this setting is used for this printout only. Subsequent printouts revert to the default set in Designer options. For a description of how to set Designer options, see “Specifying Job Dependencies” on page 5-43.



# 19

## Importing, Exporting, and Packaging Jobs

There are two situations where you may need to move or deploy DataStage components to another DataStage system:

- To continue the development of a job. In this case, you must ensure that all the components used by the job are copied to the new system. This is done by using **Export**. The components are installed into the new Repository using **Import**.
- To distribute an executable job or a plug-in. In this case, you must package the components you want to deploy using the Packager Wizard. End users install the packaged job or component using the installation program described in *DataStage Administrator's Guide*.

This chapter describes how to use the DataStage Manager to perform these tasks.

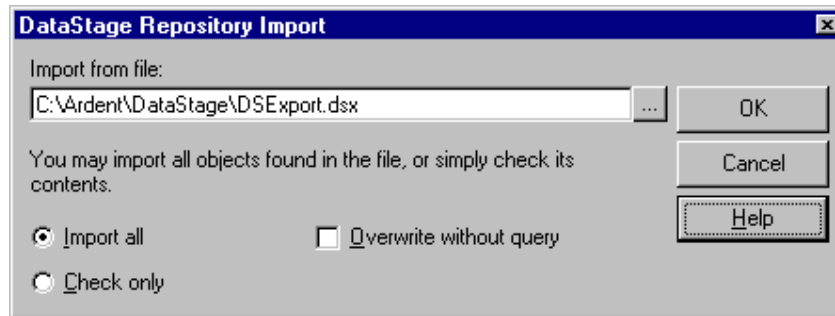
### Using Import

You can import complete projects, jobs, or job components that have been exported from another DataStage development environment. The exported components are contained in a text file. You must copy this text file to a directory you can access from your local machine.

To use the **Import** option:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.

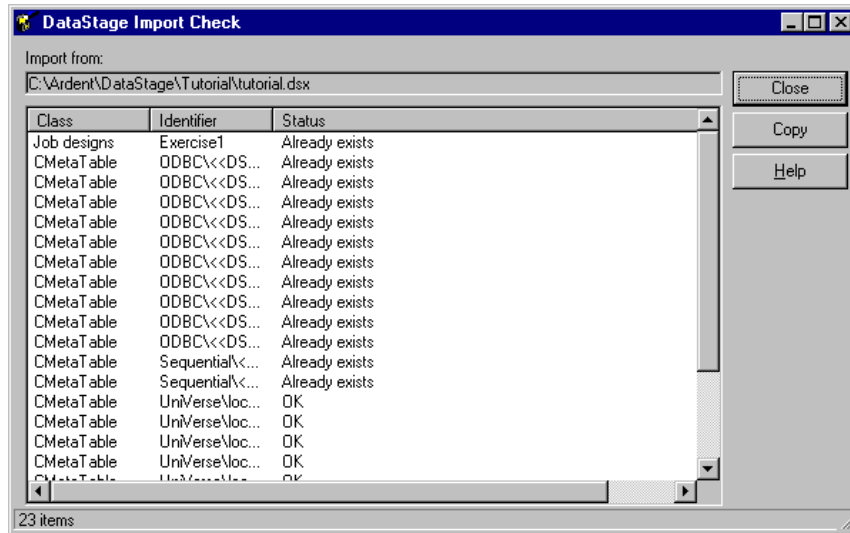
2. Choose **Tools ► Import ► DataStage Components...** . The DataStage Repository Import dialog box appears:



3. Type in the path or browse for the file to import from.
4. To import objects from the file into the Repository, click the **Import all** option button and click **OK**. During import, you will be warned if objects of the same name already exist in the Repository and asked if you want to overwrite them. If you select the **Overwrite without query** check box before importing you will not be warned, and any existing objects will automatically be overwritten.

If you import job components, they are imported into the current project in the DataStage Manager. If you import a whole project, a new project branch is created in the Repository.

5. To check the file without actually importing anything into the Repository, click the **Check only** option button, then click **OK**. Any errors encountered are displayed in a dialog box.

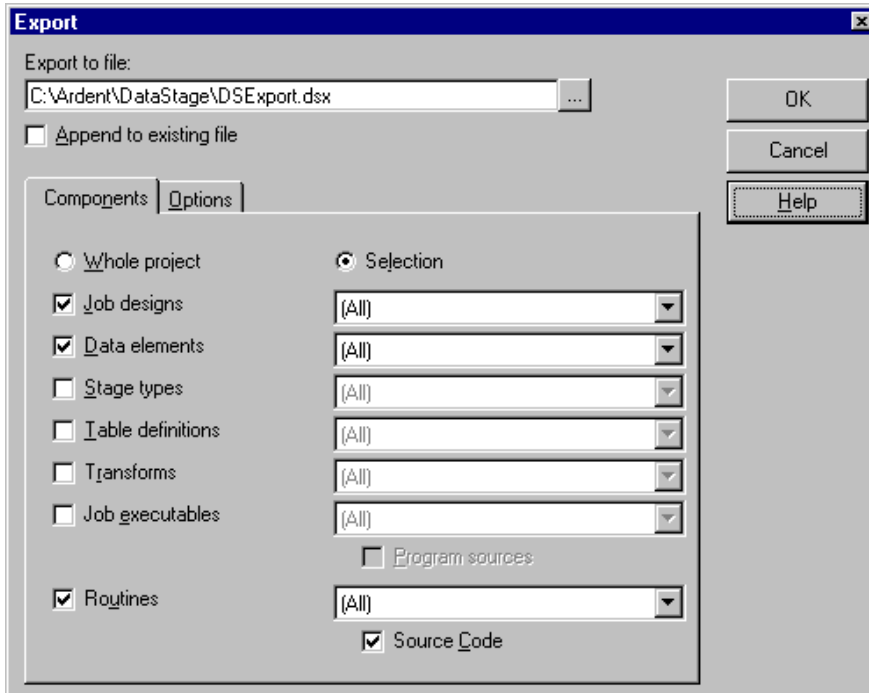


## Using Export

If you want to move a project or job components to another development environment, you must use the **Export** option in the DataStage Manager. Exported projects or components are stored in text files. You must choose what to export and the file to use.

To use the **Export** option:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools ► Export...** . The Export dialog box appears:



3. Choose the file to export to by doing one of the following:
  - Enter the directory path and file name in the **Export to file** field.
  - Click ... (browse) to search the system for an appropriate directory and file.
4. Select the **Append to existing file** check box if you want to add the exported items to the specified file. The check box is cleared by default, i.e., the exported items will overwrite any existing items in the file.
5. Choose what to export by clicking the appropriate option button:
  - **Whole project.** All the jobs, data elements, stage types, table definitions, and transforms are exported.
  - **Selection.** An individual item or a selection of items in the current project is exported.
6. If you clicked **Selection**, choose the items to export by selecting the appropriate check boxes:
  - **Job designs.** The chosen job designs are exported.

- **Data elements.** The chosen data elements are exported.
- **Stage types.** The chosen plug-in definitions are exported.
  - Note:** The **Export** option does not handle the associated plug-in DLLs. To deploy a plug-in, you must use the Packager Wizard instead. For more information, see “Using the Packager Wizard” on page 19-6.
- **Table definitions.** The chosen table definitions are exported.
- **Transforms.** The chosen transforms are exported.
- **Job executables.** The chosen job executables are exported. If you choose this option, you can also select the **Program sources** check box to export the associated source.
- **Routines.** The chosen routines are exported. Choosing this option allows you to select **Source Code** to export the associated source code.

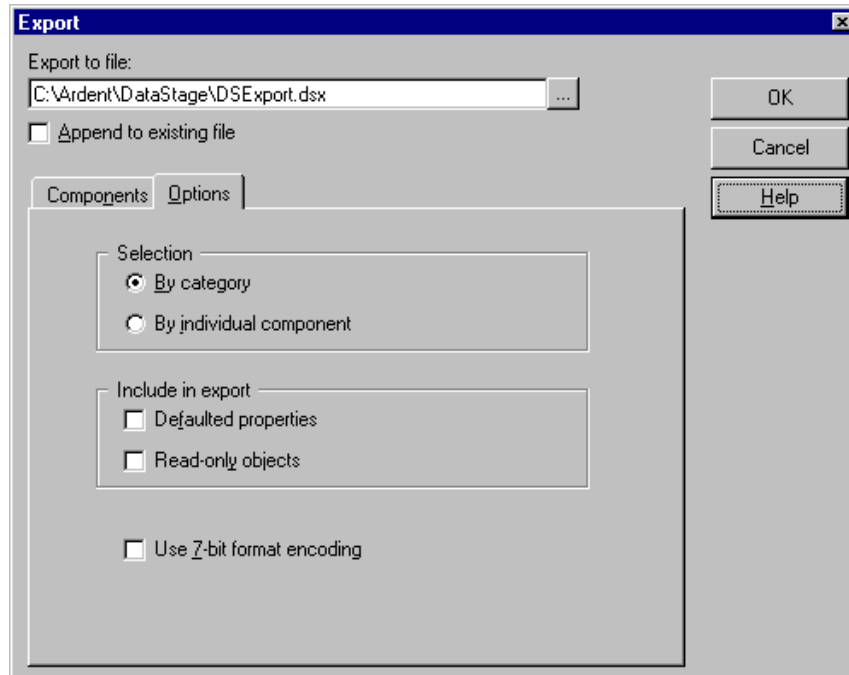
A selection is already made based on the highlighted branch in the DataStage Manager when you chose **Tools ► Export...**

By default, for objects other than job designs and job executables, you can select all objects of a particular type to be exported or select a category of object to export from the drop-down list. For example, in the case of data elements, you might choose among **All**, **Built-In**, **Built-In/Base**, **Built-In/Dates**, and **Conversion**.

It is possible to choose an individual object, but you must first click **By individual component** on the **Options** page.

7. Click the **Options** tab, then select any required options as follows:
  - **Selection - By category.** This is the default. The drop-down list boxes on the **Components** page allows you to choose between **all** and the different categories of each object in the Repository.
  - **Selection - By individual component.** This allows you to choose a single object per object type, so the drop-down list boxes on the **Components** page list all the individual objects of that type.
  - **Include in export - Defaulted properties.** Select this check box to export the properties of the items being exported. By default, properties and default values are not included in the export file.
  - **Include in export - Read-only objects.** Select this check box to export any read-only items selected in the **Components** page. By default, read-only items are not exported.

- **Use 7-bit format encoding.** Select this check box if you are likely to e-mail the exported file. The check box is cleared by default.



8. Click **OK**. The chosen items are exported to the specified text file. This file can be imported into another DataStage development environment.

## Using the Packager Wizard

If you have developed a job or plug-in, you can distribute it to other DataStage systems using the Packager Wizard. This utility is run from the Manager.

When you use the Packager Wizard, you must specify the type of package to create:

- **Job Deployment.** Contains executable job definitions (for each job in the package). The package contains information about the job itself and other dependencies such as transforms, data elements, and plug-ins.

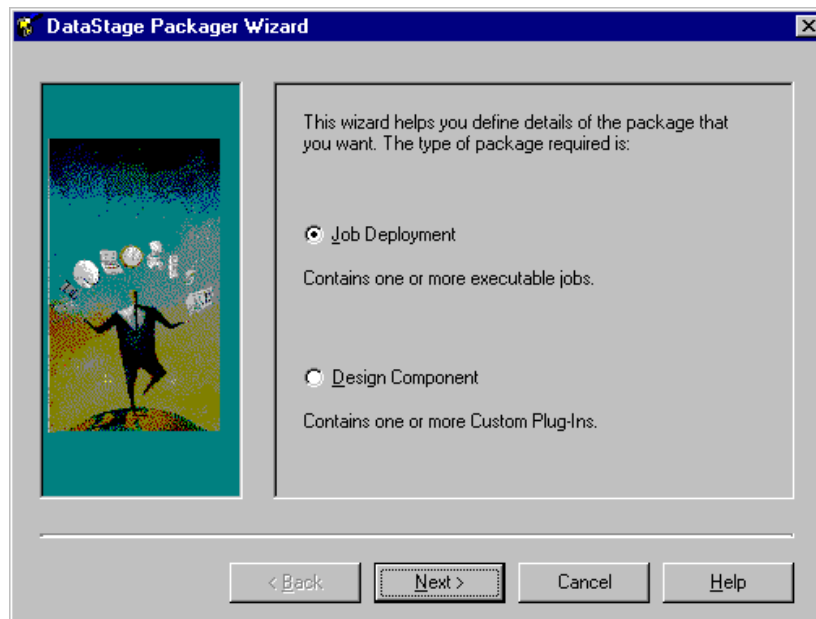


**Note:** You can only package released jobs. For more information about releasing a job, see Chapter 17, “Debugging, Compiling, and Releasing a Job.”

- **Design Component.** Contains plug-in definitions and associated DLLs.

To package a job or plug-in:

1. Start the DataStage Manager and enter your logon details in the Attach to Project dialog box. The DataStage Manager window appears.
2. Choose **Tools ► Packager Wizard...** . The DataStage Packager Wizard dialog box appears:



3. Specify the type of package you want to create by clicking **Job Deployment** or **Design Component**.
4. Click **Next>**. The updated dialog box displays the next set of prompts.
5. Enter the name of the package, a description, and a target directory on the server in the appropriate fields.

**Note:** You can use **Browse...** to search the system for a suitable directory.

6. Click **Next>**.

7. Select the jobs or plug-ins to be included in the package from the list box. The content of this list box depends on the type of package you are creating. For a Job Deployment package, this list box displays all the released jobs. For a Design Component package, this list box displays all the plug-ins under the **Stage Types** branch in the Repository (except the built-in ones supplied with DataStage).
8. If you are creating a Job Deployment package, and want to include any other jobs that the main job is dependent on in the package, select the **Automatically Include Dependent Jobs** check box. The wizard checks the dependency tree to make sure all the required jobs exist and informs you of any errors.
9. Click **Next>**. The updated dialog box confirms the package name, destination, and chosen jobs or plug-ins. If you are packaging jobs and have included dependent jobs, these are shown indented in the list.
10. Click **Finish** to create the package. The package is created in the chosen directory using the specified name.

You can now distribute the package for use on other DataStage systems.

# 20

## Using MetaBrokers

MetaBrokers allow you to exchange enterprise meta data between DataStage and other data warehousing tools. For example, you can use MetaBrokers to import into DataStage table definitions that you have set up using a data modelling tool. Similarly you can export meta data from a DataStage job to a business intelligence tool to help it analyze your data warehouse.

To use MetaBrokers, you need to install MetaBrokers for any tools with which you want to exchange meta data. MetaBrokers are provided on the DataStage installation CD (although note that you require separate licenses for them).

A list of currently available MetaBrokers is in the *Read Me* file in the *MetaBrkr* directory on the CD. Instructions for installing MetaBrokers are in *DataStage Administrator's Guide*.

This chapter tells you how to use MetaBrokers from the DataStage Manager. The instructions given are generic. Instructions for using particular MetaBrokers are in technical bulletins included on the installation CD in the relevant MetaBroker directory.

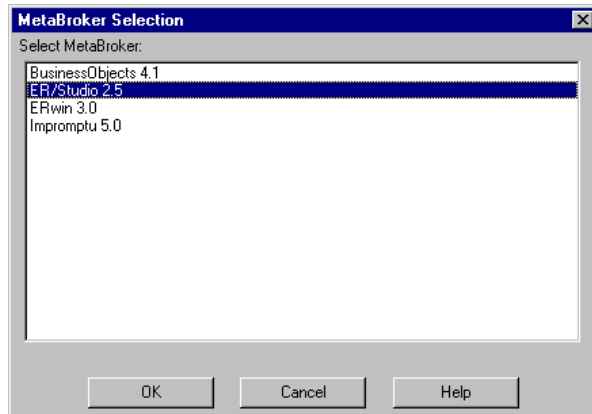
The MetaBrokers allow you to either import meta data into DataStage, or to export it from DataStage.

### Importing Meta Data

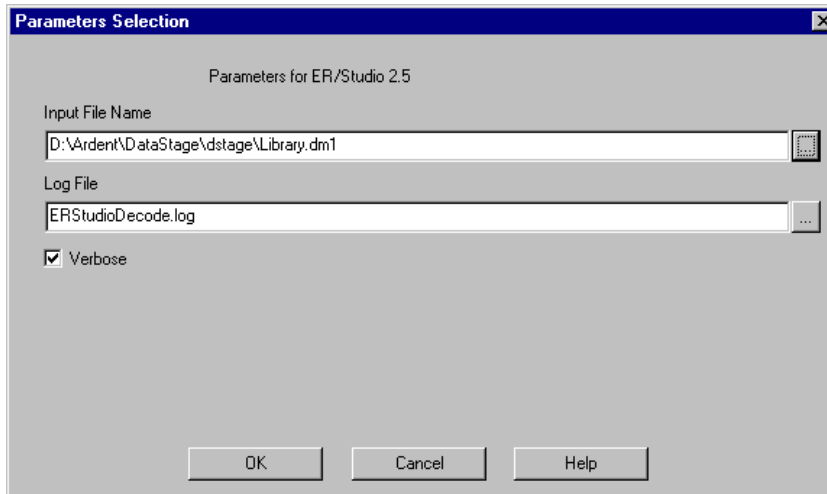
Importing meta data into DataStage has two main phases. After you have specified the tool from which you are importing and the source for the import meta data, the MetaBroker for that tool extracts the meta data from the source. This is phase one. The MetaBroker then displays a list of extracted meta data from which you can select what you actually want to import into DataStage. Once you have done this, the DataStage MetaBroker imports the meta data into the DataStage Repository. This is phase two.

To use MetaBrokers to import data into DataStage:

1. Start the DataStage Manager.
2. Choose **Tools** ► **MetaBrokers** ► **Import**. The MetaBroker Selection dialog box appears:

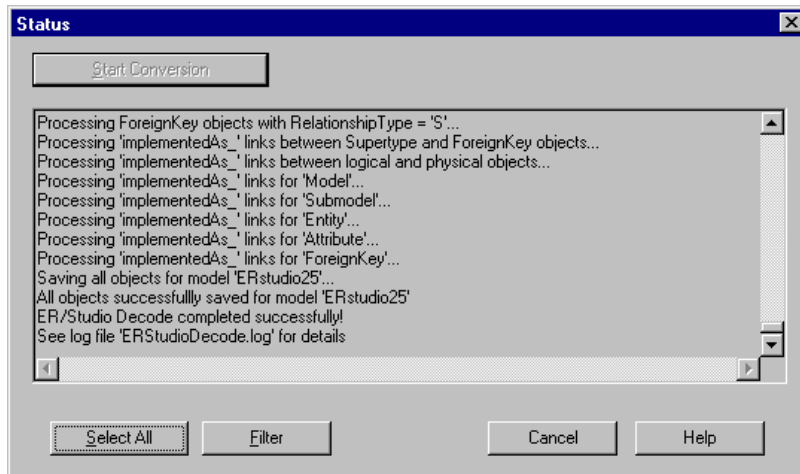


3. Select the MetaBroker for the tool from which you want to import the meta data and click **OK**. The Parameters Selection dialog box appears:

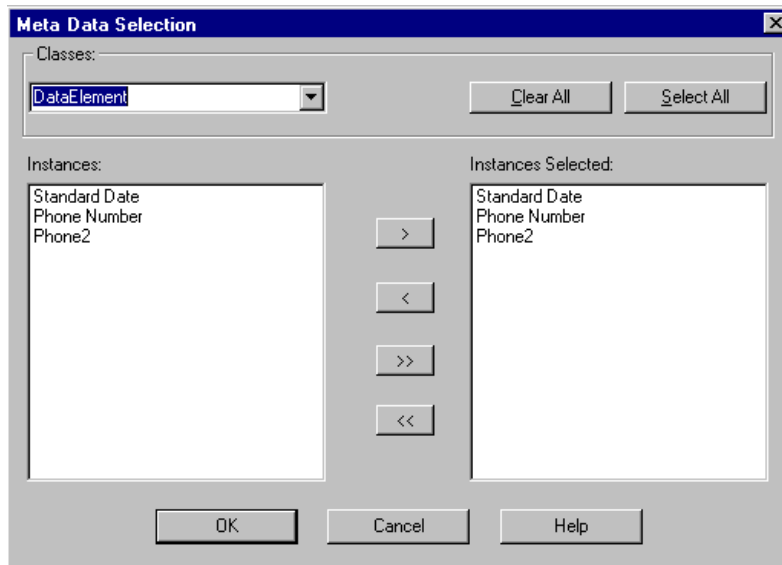


4. Specify the required parameters for the import (these vary according to MetaBroker, but typically specify data source and log file). Click **OK**. The Status dialog box appears.

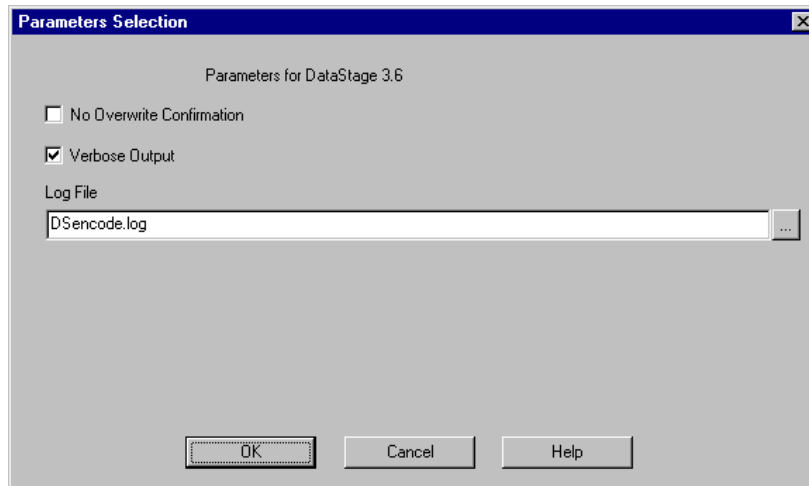
5. Click **Start Conversion** to start extracting meta data from the import source to the MetaBroker. The status of the extraction is displayed in a text window:



6. When the extraction is completed, the **Select All** and **Filter** buttons are enabled. If you want to import all the meta data, click **Select All** and continue from step 12. If you want to import selected meta data, click **Filter**. The Meta Data Selection dialog box appears. This shows the meta data that the MetaBroker has extracted from the source, and allows you to specify what you actually want to import into DataStage.

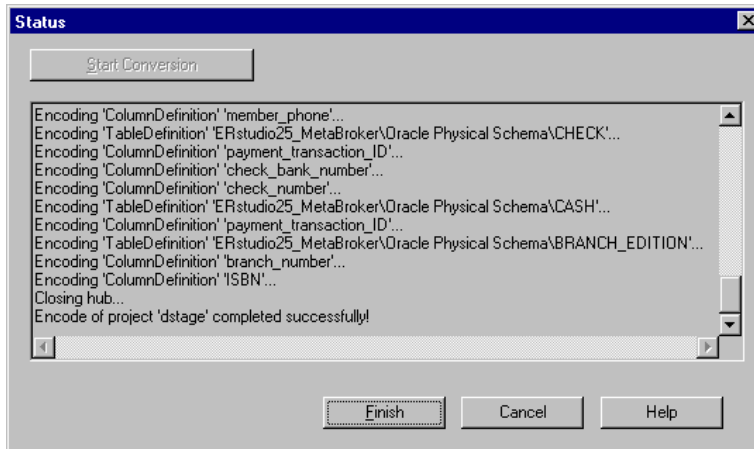


7. By default all the meta data are selected. Click **Clear All** to clear it so you can specify exactly what you want to import. (If you change your mind, click **Select All** to select all meta data again.)
8. Select the class of meta data you want to import from the **Classes** drop-down list. The instances of that class that the MetaBroker has extracted appear in the **Instances** list.
9. If you want to import all instances of that class, click >>. All the instances appear in the **Instances Selected** list. Click << to deselect them and clear the list.
10. If you want to import a particular instance, select it in the **Instances** list and click > to copy it to the **Instances Selected** list. Click < if you change your mind.
11. When you are satisfied that you have selected all the meta data you want to import, click **OK**. The Parameters Selection dialog box appears:



12. Specify the required parameters for the import into DataStage. These allow you to specify whether you should be prompted to confirm overwrites to the DataStage Repository, and whether verbose output is enabled. Click **OK**. The Status dialog box appears.
13. Click **Start Conversion** to start importing data into DataStage. The DataStage MetaBroker copies the selected data into the DataStage Repository. The status

of the import is displayed in a text window. Click **Finish** when it has completed the import.



When the import is complete, you can see the meta data that you have imported under the relevant branch in the DataStage Manager. For example, data elements imported from ER/Studio appear under the **Data Elements ► ERStudio25** branch.

You may import more items than you have explicitly selected. This is because the MetaBroker ensures that data integrity is maintained. For example, if you import a single column, the table definition for the table containing that column is also imported.

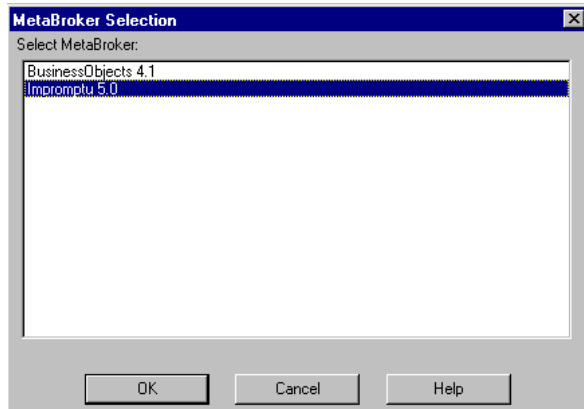
## Exporting Meta Data

Exporting meta data from DataStage to another data warehousing tool has two main phases. After you have specified the tool you want to export to, the DataStage MetaBroker extracts the meta data from the DataStage Repository. This is phase one. The MetaBroker then displays a list of extracted meta data from which you can select what you actually want to export. Once you have done this, the MetaBroker for the selected tool exports the meta data to the tool. This is phase two.

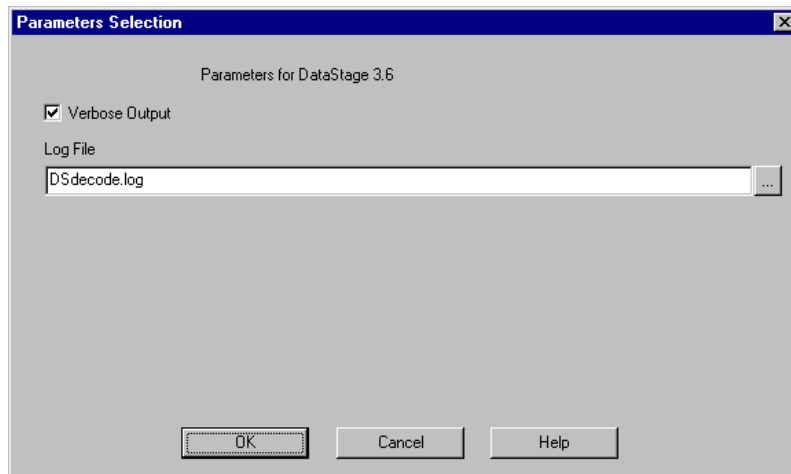
To use MetaBrokers to export data from DataStage:

1. Start the DataStage Manager.

2. Choose **Tools** ► **MetaBrokers** ► **Export**. The MetaBroker Selection dialog box appears:



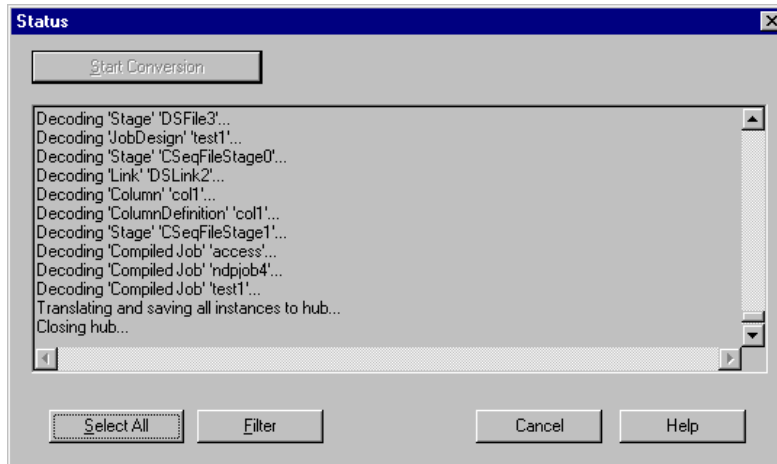
3. Select the MetaBroker for the tool to which you want to export the meta data and click **OK**. The Parameters Selection dialog box appears:



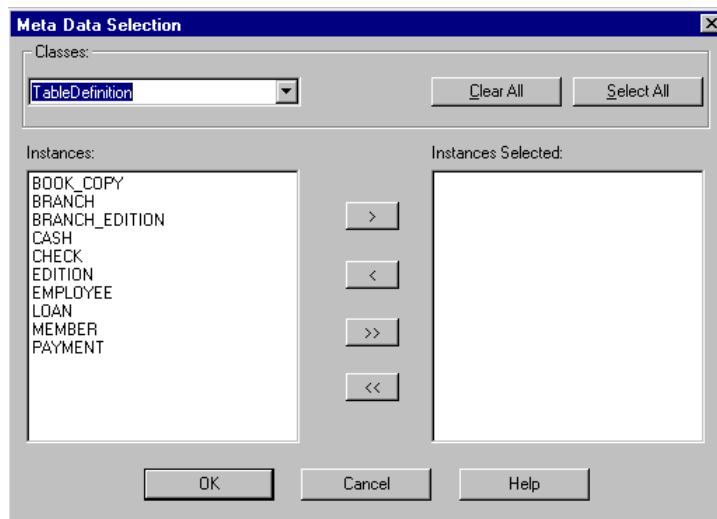
4. Specify the required parameters for the export from DataStage. These allow you to specify the name of a log file and whether verbose output is enabled. Click **OK**. The Status dialog box appears.



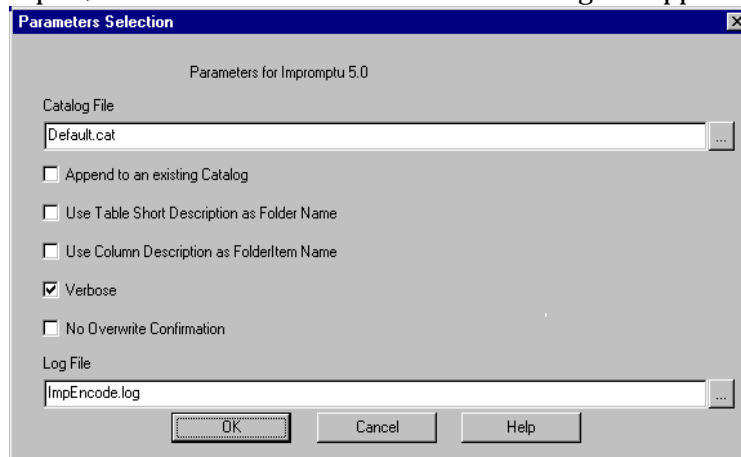
5. Click **Start Conversion** to start extracting meta data from DataStage to the MetaBroker. The status of the extraction is displayed in a text window:



6. When the extraction is completed, the **Select All** and **Filter** buttons are enabled. If you want to export all the meta data, click **Select All** and continue from step 12. If you want to export selected meta data, click **Filter**. The Meta Data Selection dialog box appears. This shows the meta data that the MetaBroker has extracted from the DataStage Repository, and allows you to specify what you actually want to export.

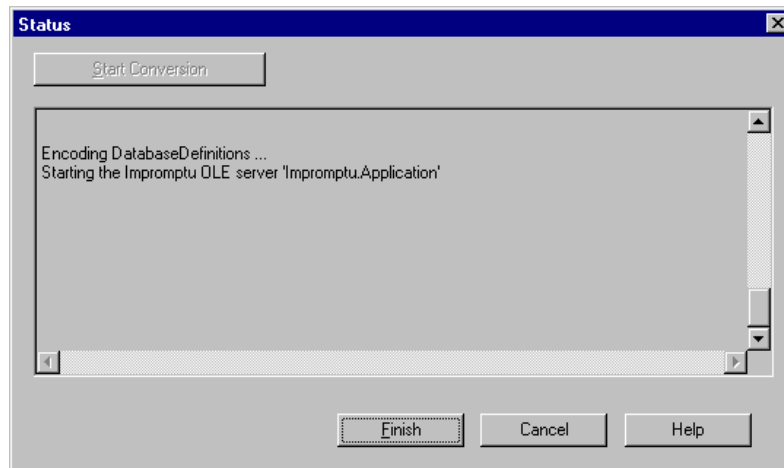


7. By default all the meta data are selected. Click **Clear All** to clear it so you can specify exactly what you want to export. (If you change your mind, click **Select All** to select all meta data again.)
8. Select the class of meta data you want to export from the **Classes** drop-down list. The instances of that class that the MetaBroker has extracted appear in the **Instances** list.
9. If you want to export all instances of that class, click >>. All the instances appear in the **Instances Selected** list. Click << to deselect them and clear the list.
10. If you want to export a particular instance, select it in the **Instances** list and click > to copy it to the **Instances Selected** list. Click < if you change your mind.
11. When you are satisfied that you have selected all the meta data you want to export, click **OK**. The Parameters Selection dialog box appears:



12. Specify the required parameters for the export into the data warehousing tool. These vary according to the tool you are exporting to, but typically include the destination for the exported data and a log file name.

13. Click **Start Conversion** to start exporting data to the tool. The status of the export is displayed in a text window. Depending on what tool you are exporting to, another application may be started to receive the exported data.



14. When the export has completed, click **Finish**.



# A

## Built-In Transforms and Routines

This appendix describes the built-in transforms and routines supplied with DataStage.

When you edit a Transformer stage, you can convert your data using one of the built-in transforms supplied with DataStage. Alternatively, you can convert your data using your own custom transforms. Custom transforms can convert data using functions or routines.

For more information about editing a Transformer stage or creating custom transforms, see Chapter 11, “Transformer Stages.” For a complete list of the supported BASIC functions, see DataStage Developer’s Help. For details on how to write a user-written routine, see Chapter 16, “Programming in DataStage.”

### **Built-In Transforms**

You can view the definitions of the built-in transforms using the DataStage Manager. For ease of reference, the transforms are listed according to their result (output) data element.

## Transforms Resulting in String

Transform	Input Type	Description
CAPITALS	String	Each word in the argument has its first character replaced with its uppercase equivalent if appropriate. Any sequence of characters between space characters is taken as a word, for example:  <code>CAPITALS("monday feb 14th") =&gt; "Monday Feb 14th"</code>
DIGITS	String	Returns a string from which all characters other than the digits 0 through 9 have been removed, for example:  <code>DIGITS("123abc456") =&gt; "123456"</code>
LETTERS	String	Returns a string from which all characters except letters have been removed, for example:  <code>LETTERS("123abc456") =&gt; "abc"</code>

## Transforms Resulting in Date

Transform	Input Type	Description
MONTH.FIRST	MONTH.TAG	Returns a numeric internal date corresponding to the first day of a month given in MONTH.TAG format (YYYY-MM), for example:  <code>MONTH.FIRST("1993-02") =&gt; 9164</code>  where 9164 is the internal representation of February 1, 1993.
MONTH.LAST	MONTH.TAG	Returns a numeric internal date corresponding to the last day of a month given in MONTH.TAG format (YYYY-MM), for example:  <code>MONTH.LAST("1993-02") =&gt; 9191</code>  where 9191 is the internal representation of February 28, 1993.

<b>Transform</b>	<b>Input Type</b>	<b>Description</b>
QUARTER.FIRST	QUARTER.TAG	<p>Returns a numeric internal date corresponding to the first day of a quarter given in QUARTER.TAG format (YYYYQn), for example:</p> <pre>QUARTER.FIRST("1993Q2") =&gt; 9133</pre> <p>where 9133 is the internal representation of January 1, 1993.</p>
QUARTER.LAST	QUARTER.TAG	<p>Returns a numeric internal date corresponding to the last day of a quarter given in QUARTER.TAG format (YYYY-MM), for example:</p> <pre>QUARTER.LAST("1993-02") =&gt; 9222</pre> <p>where 9222 is the internal representation of March 31, 1993.</p>
TIMESTAMP.TO.DATE	Timestamp	<p>Converts Timestamp format (YYYY-MM-DD HH:MM:SS) to Internal Date format, for example:</p> <pre>TIMESTAMP.TO.DATE("1996-12-05 13:46:21") =&gt; "10567"</pre>
TAG.TO.DATE	DATE.TAG	<p>Converts a string in format YYYY-MM-DD to a numeric internal date, for example:</p> <pre>TAG.TO.DATE("1993-02-14") =&gt; 9177</pre>
WEEK.FIRST	WEEK.TAG	<p>Returns a numeric internal date corresponding to the first day (Monday) of a week given in WEEK.TAG format (YYYYWnn), for example:</p> <pre>WEEK.FIRST("1993W06") =&gt; 9171</pre> <p>where 9171 is the internal representation of February 8, 1993.</p>

<b>Transform</b>	<b>Input Type</b>	<b>Description</b>
WEEK.LAST	WEEK.TAG	<p>Returns a numeric internal date corresponding to the last day (Sunday) of a week given in WEEK.TAG format (YYYYWnn), for example:</p> <pre>WEEK.LAST("1993W06") =&gt; 9177</pre> <p>where 9177 is the internal representation of February 14, 1993.</p>
YEAR.FIRST	YEAR.TAG	<p>Returns a numeric internal date corresponding to the first day of a year given in YEAR.TAG format (YYYY), for example:</p> <pre>YEAR.FIRST("1993") =&gt; 9133</pre> <p>where 9133 is the internal representation of January 1, 1993.</p>
YEAR.LAST	YEAR.TAG	<p>Returns a numeric internal date corresponding to the last day of a year given in YEAR.TAG format (YYYY), for example:</p> <pre>YEAR.LAST("1993") =&gt; 9497</pre> <p>where 9497 is the internal representation of December 31, 1993.</p>

## Transform Resulting in Time

<b>Transform</b>	<b>Input Type</b>	<b>Description</b>
TIMESTAMP.TO.TIME	Timestamp	<p>Converts TIMESTAMP format (YYYY-MM-DD) to internal time format. For example:</p> <pre>TIMESTAMP.TO.TIME("1996-12-05 13:46:21") =&gt; "49581"</pre> <p>where 49581 is the internal representation of December 5 1996, 1.46 p.m. and 21 seconds.</p>



## Transform Resulting in **TIMESTAMP**

<b>Transform</b>	<b>Input Type</b>	<b>Description</b>
TIMESTAMP	Date	Converts internal date format to <b>TIMESTAMP</b> format ( <i>YYYY-MM-DD HH:MM:SS</i> ). For example:  <pre>TIMESTAMP("10567") =&gt; "1996-12-05 00:00:00"</pre> where 10567 is the internal representation of December 5 1996.

## Transform Resulting in **DATE.TAG**

<b>Transform</b>	<b>Input Type</b>	<b>Description</b>
DATE.TAG	Date	Converts a numeric internal date to a string in <b>DATE.TAG</b> format ( <i>YYYY-MM-DD</i> ), for example:  <pre>DATE.TAG(9177) =&gt; "1993-02-14"</pre>

## Transforms Resulting in **WEEK.TAG**

<b>Transform</b>	<b>Input Type</b>	<b>Description</b>
TAG.TO.WEEK	DATE.TAG	Converts a string in <b>DATE.TAG</b> format ( <i>YYYY-MM-DD</i> ) to <b>WEEK.TAG</b> format ( <i>YYYYWnn</i> ), for example:  <pre>TAG.TO.WEEK("1993-02-14") =&gt; "1993W06"</pre>
WEEK.TAG	Date	Converts a date in internal date format to a <b>WEEK.TAG</b> string ( <i>YYYYWnn</i> ), for example:  <pre>WEEK.TAG(9177) =&gt; "1993W06"</pre>

## Transforms Resulting in MONTH.TAG

Transform	Input Type	Description
MONTH.TAG	Date	<p>Converts a numeric internal date to a string in MONTH.TAG format (YYYY-MM), for example:</p> <p>MONTH.TAG(9177) =&gt; "1993-02"</p>
TAG.TO.MONTH	DATE.TAG	<p>Converts a string in DATE.TAG format (YYYY-MM-DD) to MONTH.TAG format (YYYY-MM), for example:</p> <p>TAG.TO.MONTH("1993-02014") =&gt; "1993-02"</p>

## Transforms Resulting in QUARTER.TAG

Transform	Input Type	Description
QUARTER.TAG	Date	<p>Converts a numeric internal date to a string in QUARTER.TAG format (YYYYQn), for example:</p> <p>QUARTER.TAG(9177) =&gt; "1993Q2"</p>
TAG.TO.QUARTER	DATE.TAG	<p>Converts a string in DATE.TAG format (YYYY-MM-DD) to QUARTER.TAG format (YYYYQn), for example:</p> <p>TAG.TO.QUARTER("1993-02-14") =&gt; "1993Q2"</p>
MONTH.TO.QUARTER	MONTH.TAG	<p>Converts a string in MONTH.TAG format (YYYY-MM) to QUARTER.TAG format (YYYYQn), for example:</p> <p>MONTH.TO.QUARTER("1993-02") =&gt; "1993Q1"</p>

## Transforms Resulting in YEAR.TAG

Transform	Input Value	Description
YEAR.TAG	Date	Converts a date in internal Date format to YEAR.TAG format (YYYY), for example:  YEAR.TAG(9177) => "1993"
TAG.TO.YEAR	DATE.TAG	Converts a string in DATE.TAG format (YYYY-MM-DD) to YEAR.TAG format (YYYY), for example:  TAG.TO.YEAR("1993-02-14") => "1993"
MONTH.TO.YEAR	MONTH.TAG	Converts a string in MONTH.TAG format (YYYY-MM) to YEAR.TAG format (YYYY), for example:  MONTH.TO.YEAR("1993-02") => "1993"
QUARTER.TO.YEAR	QUARTER.TAG	Converts a string in QUARTER.TAG format (YYYYQn) to YEAR.TAG format (YYYY), for example:  QUARTER.TO.YEAR("1993Q2") => "1993"

## Built-In Routines

There are two types of routines supplied with DataStage:

- Built-in before/after subroutines. These routines are stored under the **Routines > Built-In > Before/After** branch in the Repository. They are compiled and ready for use as a before-stage or after-stage subroutine or as a before-job or after-job routine.
- Examples of transform functions. These routines are stored under the **Routines > Examples > Functions** branch in the Repository and are used by the built-in transforms supplied with DataStage. You can copy these routines and use them as a basis for your own user-written transform functions.

You can view the definitions of these routines using the DataStage Manager, but you cannot edit them.

## Built-In Before/After Subroutines

There are three built-in before/after subroutines supplied with DataStage:

- **ExecDOS.** This routine executes a command via an MS-DOS shell. The command executed is specified in the routine's input argument.
- **ExecTCL.** This routine executes a command via a UniVerse shell. The command executed is specified in the routine's input argument.
- **ExecSH.** This routine executes a command via a UNIX Korn shell.

These routines appear in the drop-down list box of available built-in routines when you edit the **Before-stage subroutine** or **After-stage subroutine** fields in an Aggregator, Transformer, or active plug-in stage, or the **Before-job subroutine** or **After-job subroutine** fields in the Job Properties dialog box.

You can also copy these routines and use the code as a basis for your own before/after subroutines.

If NLS is enabled, you should be aware of any mapping requirements when using ExecDOS and ExecSH routines. If these routines use data in particular character sets, then it is your responsibility to map the data to or from UNICODE.

## Example Transform Functions

These are the example transform functions supplied with DataStage:

- **ConvertMonth.** Transforms a MONTH.TAG input. The result depends on the value for the second argument:
  - F (the first day of the month) produces a DATE.TAG.
  - L (the last day of the month) produces a DATE.TAG.
  - Q (the quarter containing the month) produces a QUARTER.TAG.
  - Y (the year containing the month) produces a YEAR.TAG.
- **ConvertQuarter.** Transforms a QUARTER.TAG input. The result depends on the value for the second argument:
  - F (the first day of the month) produces a DATE.TAG.
  - L (the last day of the month) produces a DATE.TAG.
  - Y (the year containing the month) produces a YEAR.TAG.
- **ConvertTag.** Transforms a DATE.TAG input. The result depends on the value for the second argument:
  - I (internal day number) produces a Date.
  - W (the week containing the date) produces a WEEK.TAG.

- M (the month containing the date) produces a MONTH.TAG.
- Q (the quarter containing the date) produces a QUARTER.TAG.
- Y (the year containing the date) produces a YEAR.TAG.
- **ConvertWeek.** Transforms a WEEK.TAG input to an internal date corresponding to a specific day of the week. The result depends on the value of the second argument:
  - 0 produces a Monday.
  - 1 produces a Tuesday.
  - 2 produces a Wednesday.
  - 3 produces a Thursday.
  - 4 produces a Friday.
  - 5 produces a Saturday.
  - 6 produces a Sunday.

If the input does not appear to be a valid WEEK.TAG, an error is logged and 0 is returned.

- **ConvertYear.** Transforms a YEAR.TAG input. The result depends on the value of the second argument:
  - F (the first day of the year) produces a DATE.TAG.
  - L (the last day of year ) produces a DATE.TAG.
- **QuarterTag.** Transforms a Date input into a QUARTER.TAG string (YYYYQn).
- **Timestamp.** Transforms a timestamp (a string in the format YYYY-MM-DD HH:MM:SS) or Date input. The result depends on the value for the second argument:
  - TIMESTAMP produces a timestamp with time equal to 00:00:00 from a date.
  - DATE produces an internal date from a timestamp (time part ignored).
  - TIME produces an internal time from a timestamp (date part ignored).
- **WeekTag.** Transforms a Date input into a WEEK.TAG string (YYYYWmn).



# B

## Editing Grids

DataStage uses grids in many dialog boxes for displaying data. This system provides an easy way to view and edit tables. This appendix describes how to navigate around grids and edit the values they contain.

### Grids

The following screen shows a typical grid used in a DataStage dialog box:

	Column name	Key	SQL type	Length	Scale	Nullable	Display	Data element
▶	QUARTER	Yes	Char	6		No	6	QUARTER.TAG
■	CUSTID	Yes	Numeric	6		No	8	Number
■	TOTALSALE	No	Numeric	8	2	Yes	10	Number
*								

On the left side of the grid is a row selector column. The black triangle shows the current row. The star indicates an empty row ready for data input. The current cell is highlighted by a dotted border. The current cell is not visible if you have scrolled it out of sight.

## Navigating in the Grid

You can move around the grid by using the mouse and scroll bars, or the keyboard. Table B-1 shows the keys that are used for navigation in the grid.

**Table B-1. Keys Used in Grid Navigation**

<b>Key</b>	<b>Action</b>
Right Arrow	Move to the next cell on the right.
Left Arrow	Move to the next cell on the left.
Up Arrow	Move to the cell immediately above.
Down Arrow	Move to the cell immediately below.
Tab	Move to the next cell on the right. If the current cell is in the rightmost column, move forward to the next control on the form.
Shift-Tab	Move to the next cell on the left. If the current cell is in the leftmost column, move back to the previous control on the form.
Page Up	Scroll the page down.
Page Down	Scroll the page up.
Home	Move to the first cell in the current row.
End	Move to the last cell in the current row.
Alt-Down Arrow	In a drop-down list box, enter edit mode and move down the list.



## Editing in the Grid

You can edit the contents of the current cell in three ways:

- Start typing in the cell.
- Press the **F2** key to put the cell into edit mode.
- Choose the **Edit cell...** option from the shortcut menu.

When you start editing, the current cell highlight changes to an insertion point or a drop-down list box appears, as appropriate. Table B-2 shows the keys that are used for editing in the grid.

**Table B-2. Keys Used in Grid Editing**

<b>Key</b>	<b>Action</b>
Esc	Cancel the current edit. The grid leaves edit mode, and the cell reverts to its previous value. The focus does not move.
Enter	Accept the current edit. The grid leaves edit mode, and the cell shows the new value. A pencil symbol appears in the row selector column to indicate that the row has been modified. When the focus moves away from a modified row, the row is validated. If the data passes validation, the pencil icon is removed; otherwise, a message box is displayed, and the focus returns to the modified row.
Up Arrow	Move the selection up a drop-down list or to the cell immediately above.
Down Arrow	Move the selection down a drop-down list or to the cell immediately below.
Left Arrow	Move the insertion point to the left in the current value. When the extreme left of the value is reached, exit edit mode and move to the next cell on the left.
Right Arrow	Move the insertion point to the right in the current value. When the extreme right of the value is reached, exit edit mode and move to the next cell on the right.
Ctrl-Enter	Enter a line break in a value.

## Adding Rows

You can add a new row by entering data in the empty row indicated by the star. When you move the focus away from the row, the new data is validated. If it passes validation, it is added to the table, and a new empty row appears. Alternatively, press the **Insert** key or choose **Insert row...** from the shortcut menu, and a row is inserted with the default column name **Newn**, ready for you to edit (where *n* is an integer providing a unique **Newn** column name).

## Deleting Rows

To delete a row, click anywhere in the row you want to delete. The triangle moves to show that this is now the current row. Press the **Delete** key or choose **Delete row** from the shortcut menu. To delete multiple rows, hold down the **Ctrl** key and click in the row selector column for the rows you want to delete and press the **Delete** key or choose **Delete row** from the shortcut menu.

# C

## Troubleshooting

This appendix describes problems you may encounter with DataStage and gives solutions.

### Cannot Start DataStage Clients

Check that UniVerse is actually running on the server.

On Windows NT servers, ensure that the UniVerse Resource service, UniVerse RPC service, and UniVerse Telnet service are all started.

On UNIX servers, ensure that the RPC daemon (*unirpcd*) is started.

### Problems While Working with UniData

For more information about connecting to UniData sources, see the technical bulletin *Accessing UniVerse and UniData Databases from DataStage (74-0121)*. This is supplied with the DataStage online documentation.

### Connecting to UniData Databases

When the DataStage server is installed, a copy of the UniData API configuration file *UNIAPI.INI* is installed in the Windows directory. If, when you first attempt to connect to your UniData server, you get an error message similar to:

```
UniData Client error: call to UniOpenPos returned 45 - Client
version (11) and server version (12) are incompatible
```

then you must edit the *UNIAPI.INI* file and change the value of the `PROTOCOL` variable. In this case, change it from 11 to 12:

```
PROTOCOL = 12
```

## Importing UniData Meta Data

When importing UniData meta data, note that the server name required in the DataStage Manager dialog box is the UniData server node name, not the Object-Call (UNIAPI.INI) file entry name.

If the UniData server is on the same node as the DataStage server, then the name **localhost** can be used.

## Using the UniData Stage

The UniData stage uses the UniData Basic ITYPE function to evaluate virtual attributes and this requires that the virtual attributes are compiled. If they are not, when the link is opened an error occurs which indicates the first uncompiled I-type/virtual attribute it finds. In this instance, the solution is to compile all of the I-types/virtual attributes, not just the first one reported in the error message.

There are certain verbs that need to be available for use on the UniData database when the job containing the UniData stage is first run, or when the Data Browser is first used in the stage. These are:

- BASIC
- CATALOG
- NEWPCODE

The BP directory file must also be available.

After the initial job run or first use of the Data Browser, these can be removed or disabled if required for security reasons.

## Problems with the Documentation Tool

### Installing the Documentation Tool

You should avoid trying to install the Documentation Tool in a directory with insufficient disk space. If the install runs out of space, an error can occur which removes the Tahoma font from the system, and that can subsequently cause problems with Microsoft Word. If this situation arises, the Tahoma font can be found in the Documentation Tool install directory, from where you can reinstall it.

You should avoid running the setup program from within the Documentation Tool directory to remove or reinstall the tool. The operation will fail. Always use the main setup program on the DataStage CD.

## Using Plug-In Reports

The report for a plug-in stage type does not include information as to whether the stage supports reference links.

## Problems Running Jobs

### Job Compiles Successfully but Will Not Run

Check that your job design does not have cyclic dependencies within a sequence of active stages. This may cause your job to fail as one stage waits for another, which is in turn waiting for it.

### Job from Previous DataStage Release Will Not Run

If you run a job created using an earlier DataStage release, you may get the message:

```
Job has not been compiled with compatible compiler
```

The solution is to recompile, rerelease, and, if necessary, repack jobs under the later release of DataStage.

## Miscellaneous Problems

### Turning Grid Lines On and Off

You can turn Designer grid lines on and off from the Options dialog box, but, unlike the other options, this one does not take effect until the next time you start the Designer.

### Landscape Printing

You cannot print in landscape mode from the DataStage Manager or Director clients.

### Browsing for Directories

When browsing directories within DataStage, you may find that only local drive letters are shown. If you want to use a remote drive letter, you should type it in rather than relying on the browse feature.



# Index

## A

- account name
  - Hashed File stage 9-2
  - UniData stage 8-2
  - UniVerse stage 7-3
- ActiveX (OLE) functions 4-48
  - importing 4-49
  - programming functions 16-3, 16-9
- adding
  - Container stages 5-28
  - plug-in stages 5-13
  - stages 3-9, 5-13
- after-job subroutines 2-5, 5-32
- after-load stored procedures 14-5, 14-7
- after-stage subroutines 2-5
  - for Aggregator stages 12-2
  - for plug-in stages 13-7
  - for Transformer stages 11-5, 11-11
- aggregating data 1-3
  - using a UniVerse stage 7-13
  - using an Aggregator stage 12-7
  - using an ODBC stage 6-16
  - using COUNT function 6-17, 7-14
  - using SUM function 6-17, 7-14
- AGGREGATOR plug-in 12-9
- Aggregator stages 2-5, 5-11
  - editing 12-1
  - input data to 12-3
  - Inputs page 12-3
  - output data from 12-5
  - Outputs page 12-5
  - sorting input data 12-4
  - specifying after-stage subroutines 12-2

- specifying before-stage subroutines 12-2
- Stage page 12-2
- alternative projects 4-8
- assessing your data 4-1
- assigning data elements 3-6, 4-46
- associations 4-30
- Attach to Project dialog box 3-2, 4-2, 5-2

## B

- bad rows 15-6
- BASIC routines
  - before/after subroutines 16-10
  - copying 16-17
  - creating 16-10
  - editing 16-17
  - entering code 16-11
  - name 16-5
  - renaming 16-18
  - saving code 16-12
  - testing 16-13
  - transform functions 16-10
  - type 16-6
  - version number 16-7
  - viewing 16-17
  - writing 16-1
- BCP utility 14-1
  - running from command line 14-1
  - switches 14-1
  - unsupported switches 14-2
- BCPLoad plug-in
  - definition 14-4
  - stage properties 14-4
- BCPLoad plug-in definitions 14-4

- BCPLoad stages 2-5, 14-1
    - defining maps 14-7
    - demo 14-11
    - editing 14-6
    - overview 14-1
    - specifying job parameters 14-6
    - stored procedures 14-7
    - supported SQL data types 14-3
    - table definitions 14-3
    - using 14-5
  - before/after-subroutines
    - built-in A-7
    - creating 16-10
  - before-job subroutines 2-5, 5-32
  - before-load stored procedures 14-5, 14-7
  - before-stage subroutines 2-5
    - for Aggregator stages 12-2
    - for plug-in stages 13-7
    - for Transformer stages 11-5, 11-11
  - breakpoints 17-1
  - browsing server directories 5-23, C-3
  - built-in
    - before/after-subroutines A-7
    - data elements 2-5, 4-47
    - routines A-7
    - stages 5-11
    - transforms 2-5
  - bulk copy API 14-1
  - bulk copy program 14-1
- C**
- changing the printer setup 18-10
  - character set maps
    - and ODBC stages 6-2, 6-4
    - and plug-in stages 13-4, 13-12
    - and Sequential File stages 10-3
    - and UniData stages 8-3
    - defining 6-4, 8-3, 10-3, 13-4, 14-7, 15-4
    - specifying 5-41
  - cleaning data values 15-2
  - column definitions 2-6, 4-25
    - column name 4-25, 5-18, 12-4, 12-6
    - data element 4-25, 5-19, 12-4, 12-7
    - defining for a stage 5-18
    - deleting 4-32, 4-42, 5-20
    - editing 4-32, 4-42, 5-20
      - using the Columns grid 5-20
      - using the Edit button 5-20
    - entering 4-29
    - inserting 5-20
    - key fields 4-25, 5-18, 12-6
    - length 4-25, 5-18, 12-4, 12-7
    - loading 4-31, 5-22
    - scale factor 4-25, 5-18, 12-4, 12-7
  - columns
    - renaming by mapping 15-4
    - renaming during Orabulk stage 15-2
  - Columns grid 4-25, 5-18
  - commands, *sqlldr* 15-1, 15-5
  - compiling
    - code in BASIC routines 16-12
    - jobs 3-21, 17-1
  - connection type 6-2, 7-2
    - UniVerse 7-2
  - Container Input stage 5-12, 5-29
  - Container Output stage 5-12, 5-29
  - Container stages 2-6, 5-12, 5-28
  - containers 2-6, 5-11, 5-27
    - creating 5-28
    - editing 5-28
    - viewing 5-28
  - converting values 15-2
  - copying
    - BASIC routine definitions 16-17
    - items in the Repository 4-8
  - creating
    - BASIC routines 16-10
    - containers 5-28
    - data elements 4-44
    - data warehouses 1-2, 4-2



- executable jobs 17-1
  - items in the Repository 4-7
  - jobs 3-7, 5-8
  - stored procedure definitions 4-39
  - table definitions 4-29
  - CTLIB client library, installing 14-2
  - currency formats 5-42
  - current cell in grids B-1
  - custom transforms 2-6
    - creating 11-15
  - customizing the Tools menu 4-9
- D**
- data
    - aggregating 1-3
    - extracting 1-3
    - sources 2-7
    - transforming 1-3
  - Data Browser 3-14, 4-33, 6-7, 7-6, 8-5, 9-4, 10-7
    - definition 2-6
    - using 5-24
  - data elements
    - assigning 3-6, 4-46
    - built-in 4-47
    - creating 4-44
    - defining 4-44
    - editing 4-47
    - viewing 4-47
  - data types, SQL Server 14-3
  - data values
    - cleaning 15-2
    - validating 15-2
  - data warehouses
    - advantages of 1-4
    - creating 4-2
    - example 3-1
  - DataStage
    - client components 2-1
    - concepts 2-5
    - jobs 2-2
      - overview 2-1
      - programming in 16-1
      - projects 2-2
      - server components 2-2
      - terms 2-5
    - DataStage Administrator 2-2, 2-6
    - DataStage Designer 2-1, 2-6, 5-1
      - exiting 5-11
      - main window 5-3
      - options 5-41
      - starting 5-1
    - DataStage Designer window 5-3
      - menu bar 5-3
      - shortcut menus 5-7
      - status bar 5-6
      - tool palette 5-5
      - toolbar 5-5
    - DataStage Director 2-2, 2-6
    - DataStage Manager 2-2, 2-6, 4-2
      - exiting 4-10
      - starting 3-2, 4-2
      - using 4-2, 4-7
    - DataStage Manager window 3-3, 4-4
      - display area 4-6
      - menu bar 4-4
      - project tree 4-5
      - shortcut menus 4-6
      - title bar 4-4
      - toolbar 4-5
    - DataStage Package Installer 2-2, 2-6
    - DataStage Repository 2-2, 2-7
    - DataStage Server 2-2
    - date formats 5-42
    - DBLIB client library, installing 14-2
    - Debug Window 17-2
    - debugger 17-1
      - toolbar 5-7, 17-5
    - Define Sequential Metadata dialog
      - box 4-21
    - defining
      - character set maps 6-4, 8-3, 10-3, 13-4, 14-7, 15-4

- data elements 4-44
  - data warehouses 1-3
  - job parameters 5-34
  - locales 5-41
  - maps 5-41
  - table definitions 3-4
  - deleting
    - column definitions 4-32, 4-42, 5-20
    - items in the Repository 4-8
    - job parameters 5-36
    - links 5-17
    - stages 5-15
  - demo, BCPLoad stage 14-11
  - developing jobs 3-9, 5-1, 5-12
  - Diagram window 5-4
  - directory path
    - Hashed File stage 9-2
    - Sequential File stage 10-2
  - display area in DataStage Manager
    - window 4-6
  - documentation conventions xv
  - Documentation Tool 18-4
    - dialog box 18-5
    - toolbar 18-5
    - troubleshooting C-2
  - Documentation Tool dialog box 18-4
  - Documentation Tool toolbar 18-5
- E**
- edit mode in grids B-3
  - editing
    - Aggregator stages 12-1
    - BASIC routine definitions 16-17
    - BCPLoad stage 14-6
    - column definitions 4-32, 4-42, 5-20
      - using the Columns grid 5-20
      - using the Edit button 5-20
    - containers 5-28
    - data elements 4-47
    - grids B-1, B-3
    - job parameters 5-36
    - job properties 5-30
    - ODBC stages 6-1
    - plug-in stage properties 13-10
    - Repository items 4-7
    - stages 3-12, 5-18
    - stored procedure definitions 4-42
    - table definitions 4-32
    - Transformer stages 11-6
    - UniVerse stages 7-1
  - empty values 15-7
  - entering
    - code in BASIC routines 16-11
    - column definitions 4-29
    - SQL statements 6-7, 6-12, 7-6, 7-10
  - environment variables 15-5
  - equijoins 11-4
  - error handling 19-6, 19-7
  - errors
    - and BCPLoad stage 14-8
    - and Orabulk stage 15-6
    - and UniData stage C-1
    - compilation 17-7
  - examples
    - of projects 3-1
    - of reports 18-7
    - of routines A-7
    - of transform functions A-7
  - exiting
    - DataStage Designer 5-11
    - DataStage Manager 4-10
  - Export option, using 19-3
  - exporting
    - from the Manager 19-3
    - job components 19-3
    - jobs 19-3
    - meta data using MetaBrokers 20-5
  - Expression Editor 2-6, 5-37, 5-47, 11-12
    - configuring 5-47
  - external ActiveX (OLE) functions 4-48
  - extracting data 1-3

## F

- file formats for Sequential File stages 10-5, 10-8
- file names
  - in Hashed File stage 9-3, 9-5
  - in Sequential File stage 10-5, 10-8
  - in UniData stage 8-5, 8-6
- files
  - BCPLoad demo 14-11
  - job log 14-8
  - loading into Oracle 15-5
  - log 15-5
- Find dialog box 4-33, 16-15
- FROM clause 6-13, 7-11

## G

- generated queries
  - for ODBC input data 6-8
  - for ODBC output data 6-13
  - for UniVerse input data 7-6
  - for UniVerse output data 7-10
- grids B-1, C-3
  - adding rows in B-4
  - current cell B-1
  - deleting rows in B-4
  - editing B-1, B-3, C-3
  - keys used for navigating in B-2
  - keys used in editing B-3
  - navigating in B-2
  - row selector column B-1
- GROUP BY clause 6-13, 7-11
  - using 6-16, 7-14

## H

- Hashed File stages 2-6, 5-11
  - account name for 9-2
  - input data to 9-3
  - Inputs page 9-3
  - output data from 9-4

- Outputs page 9-4
  - specifying a file name for 9-3, 9-5
  - specifying the directory path 9-2
- Stage page 9-2
  - update action 9-4
- hashed files, importing meta data
  - from 4-17
- HAVING clause 6-13, 7-11
  - using 6-15, 7-13

## I

- Import option, using 19-1
- importing
  - external ActiveX (OLE) functions 4-49
  - into the Manager 19-1
  - job components 19-1
  - jobs 19-1
  - meta data
    - from a Sybase database table 14-3
    - from an SQL Server database table 14-3
    - using MetaBrokers 20-1
  - stored procedure definitions 4-35
  - table definitions 3-4, 4-11
    - from a hashed file 4-17
    - from a sequential file 4-20
    - from a UniData file 4-18
    - from a UniVerse table 4-15
    - from an ODBC data source 4-11
- input data
  - to Aggregator stages 12-3
  - to Hashed File stages 9-3
  - to ODBC stages 6-5
  - to plug-in stages 13-8
  - to Sequential File stages 10-4
  - to UniData stages 8-4
  - to UniVerse stages 7-4
- input links 11-4
- input parameters, specifying 4-40

- inserting column definitions 5-20
- installing
  - CTLIB client library 14-2
  - DBLIB client library 14-2
  - NetLIB client library 14-2
  - SQL Server client software 14-2
- integrity constraints 15-2

## J

- job 2-6
- job control routines 2-6, 5-38
- Job Details window 5-6
- job parameters 5-33
  - BCPLoad stages 14-6
  - defining 5-34
  - deleting 5-36
  - editing 5-36
  - using 5-36
- job properties 5-30
  - editing 5-30
  - saving 5-32
  - viewing 5-30
- jobs
  - compiling 3-21, 17-1
  - creating 3-7, 5-8
  - defining locales 5-41
  - defining maps 5-41
  - dependencies, specifying 5-43
  - developing 3-9, 5-1, 5-12
  - opening 5-9
  - overview 2-2
  - packaging 19-7
  - properties of 5-30
  - running 3-22
  - saving 5-8
  - version number 5-31

## K

- key field 4-25, 4-39, 5-18, 6-12, 7-10, 12-6

## Index-6

## L

- line terminators 10-2
- linking stages 3-10, 5-15, 5-16
- links
  - deleting 5-17
  - input 11-4
  - moving 5-16
  - multiple 5-17
  - output 11-4
  - reject 11-4
  - renaming 5-17
- loading
  - column definitions 4-31, 5-22
  - files into Oracle 15-5
- locales
  - and jobs 5-42
  - specifying 5-41
- log file 15-5

## M

- manually entering
  - stored procedure definitions 4-39
  - table definitions 4-28
- menu bar
  - in DataStage Designer window 5-3
  - in DataStage Manager window 4-4
- meta data
  - definition 2-7
  - importing for a BCPLoad stage 14-3
  - importing from a Sybase database table 14-3
  - importing from a UniData database C-2
  - importing from an SQL Server database table 14-3
- MetaBrokers
  - definition 2-7
  - exporting meta data 20-5
  - importing meta data 20-1

- Microsoft SQL Server 14-1
- monetary formats 5-42
- moving
  - links 5-16
  - stages 5-14
- multiple links 5-17
- multivalued data 4-24, 5-19

**N**

- Name Editor 5-4
- named pipes 10-2
- navigation in grids B-2
- NetLIB client library
  - configuring 14-2
  - installing 14-2
- NLS (National Language Support)
  - definition 2-7
  - overview 2-4
- NLS page
  - of the ODBC stage dialog box 6-2
  - of the Sequential File stage dialog box 3-20
  - of the Stage Type dialog box 13-4
  - of the Table Definition dialog box 3-6, 4-28
- normalization 5-25
  - definition 2-7
- null values 4-25, 4-39, 5-18, 12-4, 12-7, 15-7
- number formats 5-42

**O**

- ODBC stages 2-7, 5-12
  - connection type for 6-2
  - defining maps 6-4
  - editing 6-1
  - entering SQL statements in 6-7
  - input data to 6-5
    - using a generated query 6-8
    - using a stored procedure 6-9
  - using a user-defined SQL statement 6-8
- Inputs page 6-5
- output data from 6-10
  - using a generated query 6-13
  - using a stored procedure 6-19
  - using a user-defined SQL statement 6-17
- Outputs page 6-10
- procedure name for 6-6
- Stage page 6-2
- stored procedure parameters 6-12
- table name for 6-5, 6-11
- update action for 6-6
- viewing SQL statements in 6-7, 6-12

- opening a job 5-9
- Orabulk stages 2-7
  - defining maps 15-4
  - environment variables 15-5
  - errors 15-6
  - properties 15-3
- Oracle
  - bin* subdirectory 15-5
  - loading files into 15-5
  - sqlldr* command 15-1
- ORDER BY clause 6-13, 7-11
  - using 6-15, 7-13
- output data
  - from Aggregator stages 12-5
  - from Hashed File stages 9-4
  - from ODBC stages 6-10
  - from plug-in stages 13-9
  - from Sequential File stages 10-7
  - from UniData stages 8-5
- output links 11-4
- overview
  - of BCPLoad stage 14-1
  - of DataStage 2-1
  - of jobs 2-2
  - of NLS 2-4
  - of projects 2-2

of Transformer stage 11-1

## **P**

Packager Wizard, using 19-6

packaging

jobs 19-7

plug-ins 13-5, 19-7

parameter definitions

data element 4-39

key fields 4-39

length 4-39

scale factor 4-39

Parameters grid 4-38

Parameters page, ODBC stages 6-12

plug-in stages 2-7, 5-12

adding 5-13

BCPLoad 14-1

defining maps 13-12

editing properties 13-10

Inputs page 13-8

Orabulk 15-1

output data 13-9

output link properties for 13-10

Outputs page 13-9

properties for 13-6

specifying after-stage

subroutines 13-7

specifying before-stage

subroutines 13-7

specifying input data 13-8

specifying input properties for 13-9

Stage page 13-6

plug-ins 2-7, 13-1

BCPLoad 13-1

defining maps 13-4

installing 13-1

Orabulk 13-1

packaging 13-5, 19-7

registering 13-2

using 13-5

printer setup 18-10

printing

Documentation Tool dialog

box 18-4

example of reports 18-7

from the Designer 18-1, 18-10

landscape C-3

procedure name 6-6

programming in DataStage 16-1

project tree 4-5

projects

choosing alternative 4-8

example 3-1

overview 2-2

setting up 3-2, 4-1

properties

editing in plug-in stages 13-10

plug-in 13-6

plug-in stages 13-9, 13-10

## **R**

reference links 5-15

registering plug-in definitions 13-2

reject links 11-4

releasing a job 17-7

renaming

BASIC routines 16-18

columns 15-2, 15-4

items in the Repository 4-7

links 5-17

stages 5-14

replacing text in routine code 16-16

reporting

generating reports 18-3

Reporting Assistant dialog

box 18-1

Reporting Tool 18-1

Reporting Assistant dialog box 18-1

Reporting Tool 18-1

Repository 2-2, 2-7

Repository items

copying 4-8

- creating 4-7
- deleting 4-8
- editing 4-7
- renaming 4-7
- viewing 4-7
- Routine dialog box 16-5
  - Code page 16-8
  - Creator page 16-6
  - Dependencies page 16-8
  - General page 16-5
  - using Find 16-15
  - using Replace 16-15
- routine name 16-5
- routines
  - built-in before/after-subroutines A-7
  - examples A-7
  - writing 16-1
- row selector column B-1
- rows, bad 15-6
- running a job 3-22

## S

- saving
  - code in BASIC routines 16-12
  - job properties 5-32
  - jobs 5-8
- SELECT clause 6-13, 7-11
- Sequential File stages 2-7, 5-12
  - defining maps 10-3
  - file formats for 10-5, 10-8
  - input data 10-4
  - Inputs page 10-4
  - output data 10-7
  - Outputs page 10-7
  - specifying a file name for 10-5, 10-8
  - specifying line terminators 10-2
  - specifying the directory path for 10-2
  - Stage page 10-2

- sequential files, importing meta data from 4-20
- Server 2-2
- server directories, browsing 5-23
- setting file formats 10-5, 10-8
- setting up a project 3-2, 4-1
- shortcut menus
  - in DataStage Designer window 5-7
  - in DataStage Manager window 4-6
  - in Transformer Editor 11-3
- sort order 5-42
- source, definition 2-7
- specifying
  - Designer options 5-45
  - input parameters for stored procedures 4-40
  - job dependencies 5-43
- SQL
  - data precision 4-25, 4-39, 5-18, 12-4, 12-7
  - data scale factor 4-25, 4-39, 5-18, 12-4, 12-7
  - data type 4-25, 4-39, 5-18, 12-4, 12-7
  - display characters 4-25, 4-39, 5-18, 12-4, 12-7
- SQL Server
  - supported data types 14-3
  - unsupported BCP switches 14-2
  - unsupported data types 14-3
- SQL statements
  - entering 6-7, 7-6
  - syntax 6-13, 7-10
  - viewing 6-7, 6-12, 7-6, 7-9
- sqlldr command
  - log file 15-5
  - running under UNIX 15-5
  - specifying the control file 15-3
  - specifying the data file 15-3
- sqlldr command 15-5
- stage properties
  - BCPLoad 14-4
  - Orabulk 15-3

- Stage Type dialog box
  - and BCPLoad plug-ins 14-4
  - and plug-ins 13-6
- stages 2-7, 5-11
  - adding 3-9, 5-13
  - Aggregator 2-5, 5-11
  - BCPLoad 2-5
  - built-in 5-11
  - column definitions for 5-18
  - Container 2-6, 5-12
  - Container Input 5-12
  - Container Output 5-12
  - deleting 5-15
  - editing 3-12, 5-18
    - Aggregator 12-1
    - BCPLoad 14-6
    - Hashed File 9-1
    - ODBC 6-1
    - Orabulk 15-1
    - plug-in 13-6
    - Sequential File 10-1
    - Transformer 11-1
    - UniData 8-1
    - UniVerse 7-1
  - Hashed File 2-6, 5-11
  - linking 3-10
    - links 5-15
  - moving 5-14
  - ODBC 2-7, 5-12
  - Orabulk 2-7, 15-1
  - plug-in 2-7, 5-12
  - renaming 5-14
  - Sequential File 2-7, 5-12
  - specifying 5-12
  - Transformer 2-5, 2-8, 5-12
  - UniData 2-8, 5-12
  - UniVerse 2-8
- starting
  - DataStage Designer 5-1
  - DataStage Manager 3-2, 4-2
- status bar in DataStage Designer
  - window 5-6

- stored procedure definitions 4-34, 5-25
  - creating 4-39
  - editing 4-42
  - importing 4-35
  - manually defining 4-39
  - result set 4-40
  - viewing 4-42
- stored procedures 4-35
  - for ODBC input data 6-9
  - for ODBC output data 6-19
  - input parameters 6-12
  - using in a BCPLoad stage 14-7
- stream link 5-15
- supported BCP switches 14-4
- supported data types
  - SQL Server 14-3
  - Sybase Server 14-3
- Sybase Server 14-1
  - supported data types 14-3
  - unsupported BCP switches 14-2
  - unsupported data types 14-3

## T

- Table Definition dialog box 4-23
  - Columns page 4-25
  - for stored procedures 4-38
  - Format page 4-26
  - General page 4-24
  - NLS page 4-28
  - Parameters page 4-38
- table definitions 2-8, 4-11
  - column definitions 4-25
  - creating 4-29
  - defining 3-4
  - editing 4-32
  - from a BCPLoad stage 14-3
  - importing 3-4, 4-11
  - manually entering 4-28
  - viewing 4-32
- table name 6-5, 7-4
- terms and concepts 2-5



- testing BASIC routines 16-13
- time formats 5-42
- title bar in DataStage Manager
  - window 4-4
- tool palette 5-5
- toolbars
  - debugger 5-7, 17-5
  - Designer 5-5
  - Documentation Tool 18-5
  - Manager 4-5
  - Transformer Editor 11-2
- transform functions
  - creating 16-10
  - examples A-7
- Transformer Editor 2-8, 11-1
  - link area 11-2
  - meta data area 11-2
  - shortcut menus 11-3
  - toolbar 11-2
- Transformer stages 2-5, 2-8, 5-12, 11-1
  - basic concepts 11-4
  - editing 11-6
  - Expression Editor 11-12
  - specifying after-stage
    - subroutines 11-11
  - specifying before-stage
    - subroutines 11-11
  - transforms 11-15
- transforming data 1-3
- transforms 2-8
  - custom 2-6
  - in Transformer stage 11-15
- troubleshooting C-1
  - compilation errors 17-7
  - Documentation Tool C-2

**U**

- UNICODE 2-4
- UniData stages 2-8, 5-12
  - account name for 8-2
  - defining maps 8-3
- Inputs page 8-4
- output data from 8-5
- Outputs page 8-5
  - specifying a file name for 8-5, 8-6
- Stage page 8-2
- troubleshooting C-1
- update action 8-5
- UniVerse stages 2-8, 5-12, 7-1
  - account name for 7-3
  - and UniVerse connection 7-2
  - connection type for 7-2
  - editing 7-1
  - entering SQL statements in 7-6
  - input data to 7-4
    - using a generated query 7-6
    - using a user-defined SQL
      - statement 7-7
  - output data from
    - using a generated query 7-10
    - using a user-defined SQL
      - statement 7-15
  - Stage page 7-2
  - table name for 7-4, 7-9
  - update action for 7-5
  - viewing SQL statements in 7-6, 7-9
- UNIX line terminators 10-2
- unsupported BCP switches
  - SQL Server 14-2
  - Sybase Server 14-2
- unsupported data types
  - SQL Server 14-3
  - Sybase Server 14-3
- update action
  - in Hashed File stages 9-4
  - in ODBC stages 6-6
  - in UniData stages 8-5
  - in UniVerse stages 7-5
- user-defined SQL queries 6-12, 7-10
- user-defined SQL statements
  - for ODBC input data 6-8
  - for ODBC output data 6-17
  - for UniVerse input data 7-7

- for UniVerse output data 7-15
- using
  - BCPLoad stage 14-5
  - Container Input stage 5-29
  - Container Output stage 5-29
  - COUNT function 6-17, 7-14
  - Data Browser 5-24
  - DataStage Manager 4-2, 4-7
  - Export option 19-3
  - generated queries
    - for ODBC input data 6-8
    - for ODBC output data 6-13
    - for UniVerse input data 7-6
    - for UniVerse output data 7-10
  - GROUP BY clauses 6-16, 7-14
  - HAVING clauses 6-15, 7-13
  - Import option 19-1
  - job parameters 5-36
  - Packager Wizard 19-6
  - plug-ins 13-5
  - Print from the Designer 18-10
  - stored procedures
    - for ODBC input data 6-9
    - for ODBC output data 6-19
    - in BCPLoad stages 14-7
  - SUM function 6-17, 7-14
  - user-defined SQL statements
    - for ODBC input data 6-8
    - for ODBC output data 6-17
    - for UniVerse input data 7-7
    - for UniVerse output data 7-15
  - WHERE clauses 6-14, 7-12

## V

- validating data values 15-2
- values
  - converting 15-2
  - empty 15-7
  - null 15-7
- version number
  - for a BASIC routine 16-7

- for a job 5-31
- viewing
  - BASIC routine definitions 16-17
  - BCPLoad plug-in definitions 14-4
  - containers 5-28
  - data elements 4-47
  - job properties 5-30
  - plug-in definitions 13-3
  - Repository items 4-7
  - SQL statements 6-7, 6-12, 7-6, 7-9
  - stored procedure definitions 4-42
  - table definitions 4-32

## W

- WHERE clause 6-13, 7-11
  - using 6-14, 7-12
- Windows NT line terminators 10-2
- writing BASIC routines 16-1