

Informix DataStage

Tutorial

Version 3.6
October 1999
Part No. 000-6471

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 of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202.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

Welcome to the DataStage Tutorial	vii
Before You Begin	viii
Organization of This Manual	viii
Documentation Conventions	ix
DataStage Documentation	xi

Chapter 1. Introduction to DataStage

Projects	1-1
Jobs	1-1
Stages	1-3
National Language Support	1-3
Character Set Maps and Locales	1-4
Server Components	1-4
Client Components	1-5
The DataStage Manager	1-5
The DataStage Designer	1-5
The DataStage Director	1-6
The DataStage Administrator	1-6
DataStage Terms and Concepts	1-7

Chapter 2. Getting Started

The Data Model	2-1
The Star Schema	2-5
The Fact Table	2-5
Dimension Tables	2-6
The Time Dimension Table	2-7
Types of Data Sources and Targets	2-9
ODBC Data	2-9

Sequential File Data	2-9
Direct Access Data	2-9
Accessing Data from a DataStage Job	2-10
Installing Sample Data	2-10
Text Files	2-11
SQL Script Files	2-12
Creating Sample Database Tables	2-13
Running an SQL Script	2-13
DataStage Export File	2-16
Road Map to the Exercises	2-18

Chapter 3. Sample Job

Exercise 1: View and Run a Sample Job	3-2
Starting the DataStage Designer	3-3
The DataStage Designer Window	3-4
Opening the Sample Job	3-6
Viewing the Configuration of the Job Stages	3-7
Compiling the Job	3-11
Opening the DataStage Director	3-12
Validating and Running the Job	3-14
Summary	3-15

Chapter 4. Transforming Data

Exercise 2: Load Data into a Sequential File	4-2
Designing the Job	4-3
Configuring the Job Stages	4-6
Compiling the Job	4-9
Validating and Running the Job	4-9
Exercise 3: Load Data into a Relational Database Table	4-11
Exercise 4: Create Your Own Job	4-17
Exercise 5: Load the Time Dimension	4-17
Summary	4-22

Chapter 5. Handling Multiple Sources

Exercise 6: Use Multiple Sources and Targets	5-1
Exercise 7: Aggregate Data	5-9
Summary	5-12

Chapter 6. Using Your Own Meta Data

The DataStage Manager	6-2
Starting the DataStage Manager	6-2
The DataStage Manager Window	6-3
Exercise 8: Create Meta Data from a Relational Database Table	6-5
Exercise 9: Create a Table Definition Manually	6-7
Entering Column Definitions	6-8
Setting the File Format	6-10
Exercise 10: Use Your Meta Data in a Job	6-10
Summary	6-11

Chapter 7. Debugging Your Jobs

Exercise 11: Use the Job Design Debugger	7-1
Creating a Job	7-1
Running the Debugger	7-3
Summary	7-8

Chapter 8. Working with Multivalued Files

Exercise 12: Import the Multivalued File Definition	8-1
Creating the Sample Multivalued Source File	8-2
Creating the Meta Data	8-3
Exercise 13: Extract Data from a Multivalued File	8-4
Summary	8-8

Chapter 9. National Language Support

National Language Support in DataStage	9-1
Exercise 14: Convert from One Character Set to Another	9-2
Exercise 15: Use Per-Column Mapping	9-7
Exercise 16: Sorting Under Different Locales	9-11
Summary	9-15

Chapter 10. Additional Features

Plug-In Stages	10-1
BCPLoad Stages	10-1
Orabulk Stages	10-2

Transforms	10-2
Built-In Transforms	10-2
Custom Transforms	10-3
The Packager Wizard	10-4
Job Control Routines	10-5
Reports	10-6

Chapter 11. Summary

Main Features in DataStage	11-1
Recap of the Exercises	11-1

Appendix A. Sample Data Definitions

Index

Preface

This manual describes some of the features of the Informix DataStage tool set, and provides sample demonstrations of simple data extractions and transformations in a data warehousing setting. It is written for system administrators and application developers who want to evaluate the DataStage tools and examine some typical usage examples.

If you are unfamiliar with data warehousing concepts, please read Chapter 1 and Chapter 2 of *DataStage Developer's Guide* for an overview.

This manual is organized by task. It begins with introductory information and simple examples and progresses through more complex tasks. It is not intended to replace formal DataStage training, but rather to introduce you to the product and show you some of what it can do. The CD on the inside back cover contains the sample used in this manual and a PDF version of the manual, which you can view using the Adobe Acrobat Reader supplied with DataStage.

Welcome to the DataStage Tutorial

We are pleased that you selected our product, and to make things easier for you, we created this tutorial to take you through some simple examples of typical data mart extractions and transformations. Our demonstration introduces you to the functionality of DataStage, and shows you how easy common data warehousing tasks can be, with the right tools.

As you begin, you may find it helpful to start an Adobe Acrobat Reader session in another window, so you can refer to the DataStage documentation to see more complete coverage of some of the topics we cover. For your convenience, we reference specific sections in the DataStage documentation as we progress.

This document takes you through a demonstration of some of the features of our tool. We cover the basics of:

- Setting up and performing extractions from various data sources
- Creating field- and row-level transformations
- Creating and compiling jobs to load data into selected targets

Our goal is to help you understand our product and its capabilities, and help you make an informed choice when evaluating data mart tool sets.

We assume that you are familiar with fundamental database concepts and terminology because you are working with our product. We cover a lot of material throughout the demonstration process, and therefore will not waste your time with rudimentary explanations of things like the difference between primary and foreign keys. If your database skills are advanced, some of what is covered may seem like review. However, if you are new to databases, you may want to consult an experienced database user for assistance with some of the exercises.

Before You Begin

Once you have installed the DataStage server and client programs, you need to verify that certain other components are installed, configured, and functioning correctly before proceeding. Check the following list:

- 32-bit Open Database Connectivity (ODBC) drivers for database connections are needed.
- If your database is accessible only via a network, you need a network interface card and drivers.
- The database you intend to use for the examples (Microsoft SQL Server, Oracle, Sybase, Informix, UniVerse, and so on) needs to be running and available.
- You need access to an account in this database. If you do not currently have an account, then contact your database administrator.
- You need to verify the installation of any required client access tools or transport layers, such as SQLNet.
- You should have a data source name (DSN) configured to your target database.
- You need to verify which National Language Support (NLS) map and locales are configured as defaults.

Some of the items on this list may not apply to you. If you are unsure of the status of any of these things, please contact your system administrator for assistance.

Organization of This Manual

This manual contains the following:

Chapter 1 introduces the components of the DataStage tool set, and provides an overview of usage concepts and terminology.

Chapter 2 covers the data model and sample data, and describes the exercises in the manual.

Chapter 3 describes a sample job and introduces the DataStage Designer and Director.

Chapter 4 covers simple data extractions and transformations. It describes the use of sequential files and relational tables as data sources and targets.

Chapter 5 covers the use of multiple sources and targets, and demonstrates the use of constraints to direct output. It also introduces the Aggregator stage.

Chapter 6 describes how to create meta data using the DataStage Manager.

Chapter 7 describes how to step through a job with the job design debugger.

Chapter 8 is specifically for UniData and UniVerse users. It describes how to create meta data from a multivalued file, then how to extract data from the file and load it into a relational database table.

Chapter 9 describes National Language Support (NLS) functionality. It only applies where you have DataStage installed with NLS support.

Chapter 10 discusses additional DataStage features, including plug-in stages, built-in and custom transforms, packaging, job control routines, and report generation.

Chapter 11 summarizes the features covered and recaps the exercises.

Appendix A contains table and column definitions for sequential files, a hashed file, and the ODBC tables.

Documentation Conventions

This manual uses the following conventions:

Convention	Usage
Bold	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.

Convention	Usage
UPPERCASE	In syntax, uppercase indicates UniVerse commands, keywords, and options; BASIC statements and functions; SQL statements and keywords; and ProVerb commands. In text, uppercase also indicates UniVerse identifiers such as file names, account names, schema names, and record IDs.
<i>Italic</i>	In syntax, italic indicates information that you supply. In text, italic also indicates operating system commands and options, file names, and pathnames.
Courier	Courier indicates examples of source code and system output.
Courier Bold	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 File ➤ Exit ” means you should choose File from the menu bar, then choose Exit from the File pull-down menu.

The following 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 to DataStage

Informix DataStage provides a means of quickly creating usable data warehouses or data marts. It is an integrated set of tools for designing, developing, compiling, running, and administering applications that extract data from one or more data sources, perform complex transformations of the data, and load one or more target files or databases with the resulting data.

Solutions developed with DataStage are open and scalable; you can, for example, readily add data sources and targets, or handle increased volumes of data.

Projects

Whenever you start a DataStage client, you are prompted to attach to a DataStage project. Each complete project may contain:

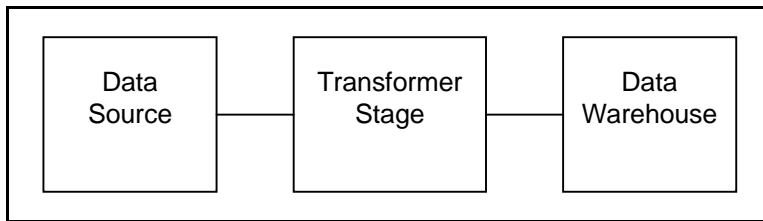
- **DataStage jobs.** A set of jobs for loading and maintaining a data warehouse.
- **Built-in components.** Predefined components used in a job.
- **User-defined components.** Customized components created using the DataStage Manager. Each user-defined component performs a specific task in a job.

Jobs

A DataStage job consists of a series of individual stages, linked together to describe the flow of data from a data source to a final data warehouse or data mart. 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 DataStage Designer.

The following diagram represents the simplest job 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?

Data properties are defined by:

- **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. DataStage provides a large library of built-in transforms.

Together, these properties determine what occurs at each stage of a DataStage job. The properties are set up project-wide and are shared by all the jobs in a project.

Compilation of a job creates an executable that is scheduled by the DataStage Director and run by the DataStage Server. 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.

Stages

A stage can be passive or active. 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.

There are two types of stage:

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

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.

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.

National Language Support

DataStage can be installed with built-in National Language Support (NLS). NLS enables DataStage to:

- Process data in a wide range of languages
- Accept data in almost 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)

This remarkable capability is made possible by DataStage using an internal character set based on the international standard UNICODE format. It invisibly translates data to and from this character set, 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.

- **Maps.** Define the character sets that the project can use.
- **Locales.** Define the local formats for dates, times, sorting order, and so on that the project can use.

The map and locale which are assigned on installation become the project defaults, and are automatically assigned to any jobs created in DataStage, but you can specify different default maps and locales for individual projects if required.

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 individual columns within a table
- For transforms and routines used to manipulate data within a stage
- For imported meta data and table definitions

The locale and character set information is 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 on that system.

Server Components

DataStage has 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, under the control of the DataStage Director, 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.

Client Components

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

- **DataStage Manager.** A user interface used to view and edit the contents of the Repository.
- **DataStage Designer.** A graphical tool used to create DataStage jobs.
- **DataStage Director.** A user interface used to validate, schedule, run, and monitor DataStage jobs.
- **DataStage Administrator.** A user interface used to set up DataStage users, control purging criteria, and install NLS maps and locales.

The exercises in this tutorial center on the clients. The server components require little interaction, although the exercises in which you use the DataStage Manager also give you the opportunity to examine the Repository.

The following sections briefly introduce the DataStage Manager, Designer, Director, and Administrator. We return to these tools during the exercises, when you learn how to use them to accomplish specific tasks. In doing so you gain some familiarity with each tool.

The DataStage Manager

The DataStage Manager is a graphical tool that enables you to view and manage the contents of the DataStage Repository.

You use it to browse, import, and edit meta data about data sources, targets, and transformations. It also allows you to define or install DataStage components such as data elements, custom plug-in stages, custom transforms, and routines.

In “Exercise 8: Create Meta Data from a Relational Database Table” and “Exercise 9: Create a Table Definition Manually” you see how easy it is to create meta data from relational database tables and from sequential files. In these exercises you the DataStage Manager and view the Repository.

For a detailed description of all the features offered by the DataStage Manager, refer to *DataStage Developer's Guide*.

The DataStage Designer

You use the DataStage Designer to build jobs by creating a visual design that models the flow and transformation of data from the data source through to the target warehouse. The DataStage Designer graphical interface lets you select stage

icons, drop them onto the Designer work area, and add links. Then, still working in the DataStage Designer, you define the required actions and processes for each stage and link. You compile your jobs from the Designer, which also provides a job debugger.

A job created with the DataStage Designer is easily scalable. This means that you can easily create a simple job, get it working, then insert further processing, additional data sources, and so on.

As you work through the exercises in this tutorial, you use the DataStage Designer to create a variety of jobs. These jobs introduce you to the input and output stages, and demonstrate some of the ways in which you can use the Transformer stage. You also do an exercise that demonstrates how to run the debugger. By the time you complete the exercises you will be able to appreciate both the power and ease of use of the DataStage Designer.

Chapter 3 describes the DataStage Designer interface, telling you enough to be able to work through the exercises. For a detailed description of all the features offered by the DataStage Designer, refer to *DataStage Developer's Guide*.

The DataStage Director

The DataStage Director enables you to run, monitor, and control jobs built in the DataStage Designer. When you run a job, the Director prompts for any run-time parameters. It lets you monitor run-time performance and events such as error conditions. You can run jobs on an ad hoc basis or schedule them.

You are introduced to the DataStage Director interface in Chapter 3, when you run a sample job installed from the tutorial CD. Each tutorial exercise in which you create a job also requires you to run that job and giving you plenty of opportunity to learn how to use the Director.

For a detailed description of all the features offered by the DataStage Director, refer to *DataStage Operator's Guide*.

The DataStage Administrator

The DataStage Administrator enables you to set up DataStage users, control the purging of the Repository, and, if NLS is installed, install and manage maps and locales. In Chapter 9 you use the DataStage Administrator to install various maps and locales which are used in the exercises to demonstrate the power of NLS.

DataStage Terms and Concepts

The following terms are used in DataStage:

Term	Description
1NF	See first normal form .
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.
column definition	Defines the columns contained in a data table. Includes the column name and the type of data contained in the column.
Container stage	A built-in stage type that represents a group of stages and links in a job design.
Data Browser	A tool used from within the DataStage Manager or DataStage Designer to view the content of a table or file.
data element	A specification that describes one item of data that can be stored in a column and the operations that can be carried out on that column.
DataStage Administrator	A tool used to configure DataStage projects and users.
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.
developer	The person designing and developing DataStage jobs.
first normal form	The name of a kind of relational database that can have only one value for each row and column position (or cell). Its abbreviation is 1NF.
hashed file	A file that uses a hashing algorithm for distributing records in one or more groups on disk.

Term	Description
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 batches	A group of jobs or separate instances of the same job (with different job parameters) that is scheduled to run.
job parameter	A variable included in the job design. A suitable value must be entered for the variable when the job is validated, scheduled, or run.
locale	Definition of local format for numbers, dates, time, currency, and other national conventions.
map	Table used internally by DataStage to enable conversion between code pages and other character set representations, and UNICODE.
meta data	Data about data. A table definition which describes the structure of the table is an example of meta data.
NF ²	See nonfirst-normal form .
NLS	National Language Support. DataStage can support the handling of data in a number of character sets.
nonfirst-normal form	The name of a kind of relational database that can have more than one value for a row and column position (or cell). Its abbreviation is NF ² .
normalization	The conversion of records in NF ² (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.

Term	Description
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 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 a data mart in a DataStage job.
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.
UNICODE	A 16-bit character set that provides unique code points for all characters in every standard character set (with room for some nonstandard characters too). UNICODE forms part of ISO 10646
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.

Term	Description
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.

2

Getting Started

As you work through the exercises in this tutorial, you create jobs that extract data, transform it, then load it into target files or tables. This chapter describes the sample data you use, and provides a road map showing where you are going and how to get there. It covers the following:

- **Data model.** An explanation of how the data fits together, starting with text files and traditional relational tables, and progressing to the target data mart environment.
- **Types of data sources and targets.** An outline of the types of data sources and targets that DataStage supports.
- **Sample data.** An overview of the sample data and instructions on how to install it.
- **Exercises.** A summary of the tasks you do.

The Data Model

The sample data is based on a very simple, extremely common business situation: a fictitious company, whose business is selling products to customers. It does not get much simpler than that!

Six tables form the basis of all the examples you work with. Each table has a simple design and contains a minimal number of rows. The focus is on the DataStage tool set and how it works, not on loading thousands of rows of data. Table 2-1 describes the first six sample tables you work with. The composite primary keys in each table are shown in bold type.

Table 2-1. Six Sample Tables

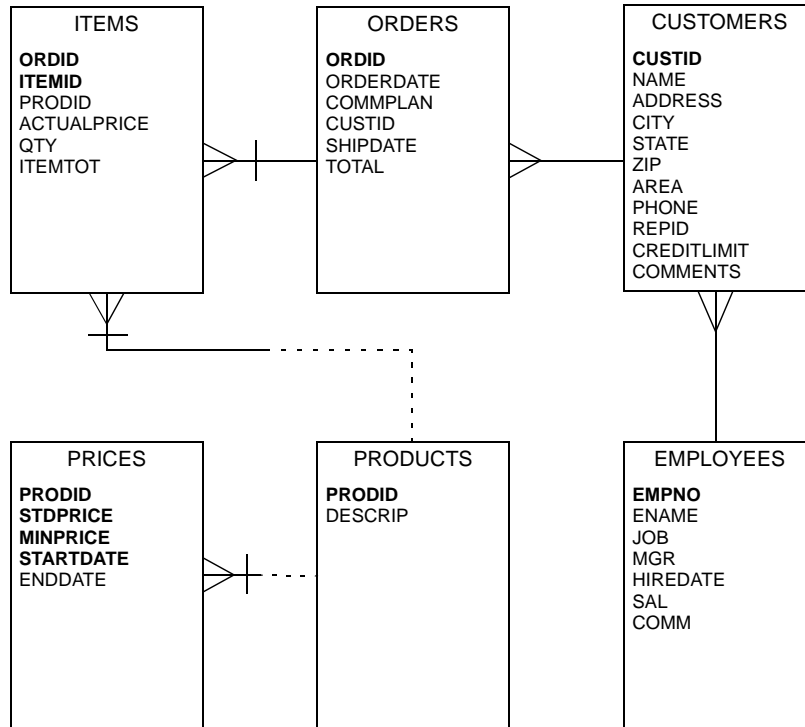
Table	Column	Description
ITEMS	ORDID	Order identifier
	ITEMID	Item number on this order
	PRODID	Product identifier
	ACTUALPRICE	Actual selling price
	QTY	Quantity ordered
	ITEMTOT	Item total for this order
ORDERS	ORDID	Order identifier
	ORDERDATE	Date order was placed
	COMMPAN	Commission plan
	CUSTID	Customer identifier
	SHIPDATE	Date order shipped
	TOTAL	Total for this order
CUSTOMERS	CUSTID	Customer ID number
	NAME	Customer name
	ADDRESS	Customer address
	CITY	Customer city
	STATE	Customer state
	ZIP	Customer postal code
	AREA	Telephone area code
	PHONE	Telephone number
	REPID	Sales representative ID
	CREDITLIMIT	Customer credit limit
	COMMENTS	Any notes about customer

Table 2-1. Six Sample Tables (Continued)

Table	Column	Description
PRICES	PRODID	Product identifier
	STDPRICE	Standard selling price
	MINPRICE	Minimum selling price
	STARTDATE	Starting date for this price
	ENDDATE	Ending date for this price
PRODUCTS	PRODID	Product identifier
	DESCRIP	Product description
EMPLOYEES	EMPNO	Employee number
	ENAME	Employee name
	JOB	Job title
	MGR	Employee supervisor
	HIREDATE	Date of hire
	SAL	Employee salary
	COMM	Employee sales commission

As you can see, the data is relatively straightforward, and consists of a small subset of what you expect to find in a company database.

In a relational database environment, the data model might be expressed like this:



For those unfamiliar with this sort of diagram, the connecting lines indicate the nature of the relationship between the tables.

For example, the line between **PRODUCTS** and **PRICES** denotes that for each product, there may be one or more prices and each price record must pertain to one and only one product. The bar through the line means that the uniqueness of **PRICES** is determined in part by the related record in the **PRODUCTS** table.

So that's what the model looks like from a traditional relational perspective. In the exercises, you migrate data from a simulated operational environment—the traditional relational model you have just seen—into a data mart environment.

The Star Schema

In data warehousing environments, the relational model is usually transformed or coalesced into a *star schema*.

The star schema model is named for its appearance. It resembles a star, with rays emanating from the center. A star schema is used to simplify report production, and increase performance of ad hoc queries against the data mart. The performance advantage created by a correctly designed star schema is tremendous.

There are two significant differences between a typical relational model and a star schema model. They are known as the *fact table* and the *dimension tables*.

The Fact Table

The chief feature of a star schema is the table at the center, called the fact table.

The fact table does not follow any of the conventional rules pertaining to data normalization. It is denormalized on purpose, to enhance query response times.

The table typically contains the records you most want to explore, usually with ad hoc queries. The records in the fact table are often referred to as *events*, due to the time-variant nature of a data mart.

You track which products customers order over time. You do not store the total of each order in the fact table (you add a summary table for this later), but instead the quantity of each product ordered, and the actual selling price for the product at that time.

In order to establish uniqueness, the primary key for the fact table is a composite of all the columns except QTY, TOTALSALE, and SHIPDATE.

The fact table used in the tutorial is named FACTS. It looks like this:

FACTS
ORDERDATE
EMPNO
CUSTID
ORDID
PRODID
QTY
TOTALSALE
SHIPDATE

In addition to the fact table, you create a table to hold *rejects*, or rows which you do not want to include in the fact table. It is created only for the purpose of demonstrating multiple targets in one of the exercises.

The name of this table is REJECTS, and its structure matches that of the fact table.

Dimension Tables

You can think of dimension tables as “spokes” on a wheel, with the fact table forming the hub, or center. Nearly all of the information in a typical fact table is also present in one or more dimension tables.

Dimension tables allow for quick browsing of specific categories of information. This is particularly useful when you want to narrow the selection criteria of your query.

The primary keys of each of the dimension tables are chained together to form the composite primary key of the fact table.

Note: In our example, ORDERDATE references records in the time dimension. It is viewed as belonging to the time dimension, and is one of the components of the primary key of the fact table.

The following table lists the columns and the dimensions they belong to:

Composite Key Column	Dimension
ORDERDATE	Time dimension
EMPNO	Employee dimension
CUSTID	Customer dimension
ORDID	Order dimension
PRODID	Product dimension

There is a special dimension table, which must be created manually. It does not already exist in this operational data, but it is crucial, because it allows control over the date ranges of the data you want your query to return. This table is called the *time dimension table*.

The Time Dimension Table

The time dimension table, TIME, contains many rows of dates, and almost always has additional columns to allow grouping of events by broader date ranges.

In these examples, you use a very simple time dimension table that looks like this:

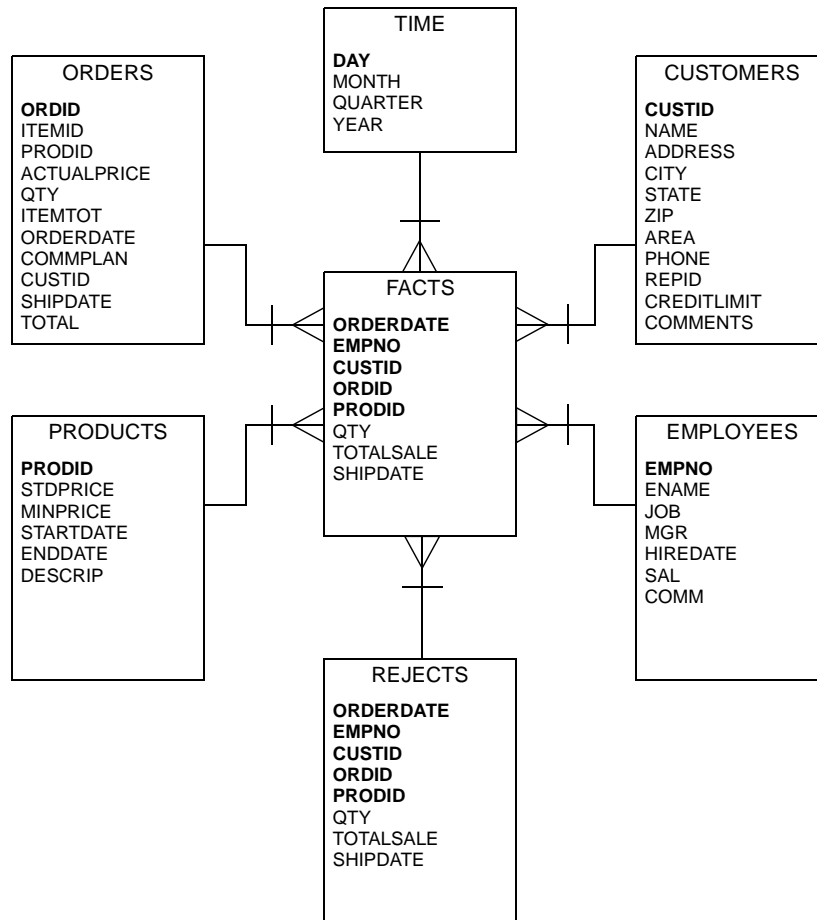
Table	Column	Description
TIME	DAY	A date including day, month, and year
	MONTH	The month to which the date belongs
	QUARTER	The fiscal quarter to which the date belongs
	YEAR	The year to which the date belongs

The time dimension table must be built manually, and must contain every possible date you could encounter in your data. For each date, you must supply the month, the quarter, and the year.

Once this is accomplished, the time dimension allows you to quickly group data by month, quarter or year, which is useful for reporting, and much more efficient than selecting every date individually, or using an expression of “greater than” one date and “less than” another.

Note: For UniVerse users, the TIME table is called TIME_ because TIME is a reserved word in UniVerse.

This is the star schema you use for the exercises. Columns shown in bold type are primary keys.



Note: The data mart model is oversimplified for demonstration purposes. We do not recommend you follow our example when modelling your data, although it is perfectly adequate for the exercises in this tutorial.

Types of Data Sources and Targets

Throughout the exercises in this manual, you extract data from one or more data sources, and load it into one or more targets. Let's look at the types of data sources and targets that DataStage supports.

DataStage considers data to belong to one of the following categories:

- ODBC data
- Sequential file data
- Direct access data

During the exercises you use Sequential File stages and, depending on your database, UniVerse, UniData, Hashed File, or ODBC stages.

ODBC Data

ODBC data is defined as any data source that can be accessed through the Open Database Connectivity API. This is accomplished by means of a database-specific ODBC driver.

The list of databases for which ODBC drivers are available is quite large, giving DataStage the ability to handle data from a wide variety of databases.

Note: DataStage requires that you use 32-bit ODBC drivers. It does not work with older 16-bit ODBC drivers. If you are unsure about which drivers you are using, contact your system administrator.

Sequential File Data

Sequential file data is also known as text files or flat files. These can be of fixed-length or delimited format. As long as you tell DataStage what you are using for a delimiter, it can be whatever you choose.

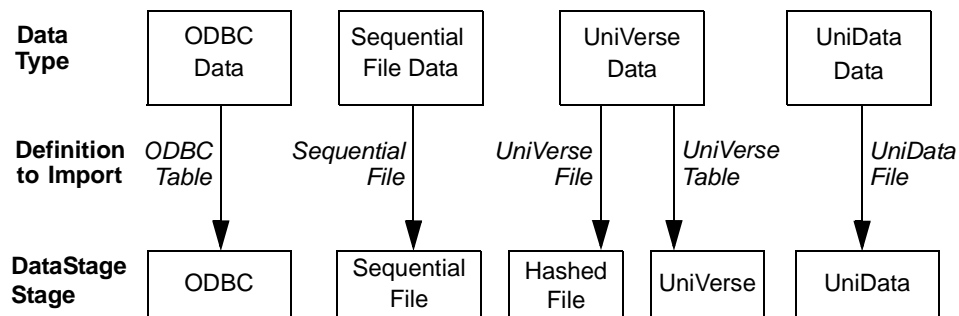
Sequential files can exist on any drive, either local or mapped to the Windows NT server.

Direct Access Data

Direct access data is accessed directly without using SQL. UniVerse hashed files and UniData databases are examples of direct access data. In this tutorial, you are shown how to access UniVerse data in table form using SQL scripts and data in a UniVerse hashed file. The tutorial also shows how to extract data from a UniData file.

Accessing Data from a DataStage Job

The following figure shows the relationship between data types, the type of stage you use to access that data in a DataStage job, and the type of file/table definition you use with that stage. UniVerse data, for example, can be in the form of a table or a hashed file; for each form you use a different type of stage and import a different type of definition.



Installing Sample Data

Before you install the sample data for the tutorial, you must have installed the DataStage Developer's Edition. You should also have verified that certain other components are installed, configured, and functioning correctly before you proceed. These are listed in "Before You Begin" on page viii.

The sequence of steps for installing and configuring the sample data is as follows:

1. Install sample data from the tutorial CD.
2. Check the database account for existing tables.
3. Run SQL scripts to create database tables.
4. Import the DataStage export file containing meta data for tables.

To install sample data for the tutorial:

1. Run *install.exe* from the DataStage 3.6 tutorial CD.
The DataStage Tutorial setup program is launched.
2. Follow the on-screen instructions in the installation wizard.

Note: During the installation, you are prompted to choose a DSN. This information should have been supplied by your database administrator. However, if you do not recognize any of the DSNs in the **Current DSNs** list box, then select the **Use UniVerse environment** check box. Remember to follow the specific “for UniVerse users” instructions during the exercises.

When you have entered the information required by the setup program, it automatically copies the following files into the specified destination directory (by default `c:\Ardent\DataStage\Tutorial`):

- Text files (also known as sequential files or flat files)
- SQL Data Definition Language (DDL) script files
- A DataStage export file, *tutorial.dsx*

Text Files

Each of the text files is named according to the data it contains. The installation includes the following text files:

- ITEMS.TXT
- PRICES.TXT
- MVPRICES.TXT
- TIME.TXT
- EBCDPROD.TXT
- PERCOL.TXT
- COLLATE.TXT

You can view these files by opening them with any text editor such as Notepad or WordPad. If you do so, notice that the ITEMS.TXT file contains rows of a fixed length, while the PRICES.TXT file is delimited with commas. The exercises demonstrate the ease with which DataStage handles either type. The EBCDPROD.TXT and PERCOL.TXT files might look rather strange if viewed in this way. They are used for the NLS-specific exercises. EBCDPROD.TXT uses the EBCDIC character set, and part of PERCOL.TXT is in Japanese.

SQL Script Files

The SQL scripts create the database tables that you use in the tutorial exercises. The following table lists the SQL script files and summarizes the purpose of each:

Database	Script Name	Description
Informix	INF_BLD.SQL	Creates example data in an Informix database.
	INF_CLN.SQL	Removes the example data from the Informix database after you have completed the tutorial exercises.
Microsoft SQL Server	MSS_BLD.SQL	Creates example data in a Microsoft SQL Server database.
	MSS_CLN.SQL	Removes the example data from the Microsoft SQL Server database after you have completed the tutorial exercises.
Oracle 8	ORA_BLD.SQL	Creates example data in an Oracle database.
	ORA_CLN.SQL	Removes example data from the Oracle database after you have completed the tutorial exercises.
Sybase	SYB_BLD.SQL	Creates example data in a Sybase database.
	SYB_CLN.SQL	Removes the example data from the Sybase database after you have completed the tutorial exercises.
UniVerse	UNI_BLD.SQL	Creates example data in a UniVerse database.
	UNI_CLN.SQL	Removes the example data from the UniVerse database after you have completed the tutorial exercises.

The script you need depends on the database you are using. “Running an SQL Script” on page 2-13 has guidelines for running the appropriate script on all the databases listed above.

You can view the content of any of the script files by using a word processor or a text editor such as Notepad. The script files are held in the destination directory you specify when you run the tutorial setup program. By default this is *c:\Ardent\DataStage\Tutorial*.

Creating Sample Database Tables

Now that you have installed the SQL scripts from the tutorial CD into the *Ardent\DataStage\Tutorial* directory, you can start up an SQL environment and execute the scripts to create sample data mart tables.

For the purposes of the exercises, it is convenient if you can use a “clean” database account, that is, one that does not contain any other tables in its catalog. If this is not possible, you *must* check the following list of table names, and ensure that the account you are using *does not* already contain tables with these names:

CUSTOMERS
EMPLOYEES
FACTS
ITEMS
MYPRICES
ORDERS
PRICES
PRODUCTS
Q_SALES
REJECTS
TIME (or TIME_ if you are a UniVerse user)

CAUTION: If you find even one table in your catalog that has the same name as a table in the list, you cannot proceed with the exercises. Doing so destroys your existing tables. You need to contact your database administrator to arrange the use of a clean database.

After you have created the sample tables, you import a DataStage export file that contains meta data for the tables.

Running an SQL Script

The SQL script files are intended to be run from within your SQL database environment. Depending on what relational database you are running, the way you get to your SQL environment may be different. This section has guidelines for starting up SQL environments on these databases and running the appropriate script to create sample tables. Note though that it may be necessary for you to adjust the syntax slightly to meet the needs of your database and SQL environment. If you are not sure how to do this, contact your database administrator for assistance.

The following lists some common relational database SQL environments, with a reference to the page that has guidance on running the tutorial script in that environment:

Relational Database	SQL environment provided by...	Described on...
Informix	DbAccess or ISQL	Page 2-14
Microsoft SQL Server	ISQL/w	Page 2-14
Oracle 8	SQL*Plus	Page 2-15
Sybase	ISQL	Page 2-15
UniVerse	Universe/SQL	Page 2-15

If you do not recognize any of the above databases, and do not have them on your system, you should use the UniVerse environment. DataStage comes with a UniVerse database, so you automatically have support for it on your system. DataStage views UniVerse data as a special type of ODBC data, and some of the tutorial details are slightly different when you use UniVerse, but you are told whenever this is the case.

For Informix. At the command line (DOS prompt), enter the following:

```
Dbaccess databasename inf_bld.sql > build.log
```

This directs the output of the script to a file, which you can then view to verify the successful completion of the script.

For Microsoft SQL Server. On Windows NT 4.0 or Windows 98, choose the following:

Start ► Programs ► Microsoft SQL Server 6.5 ► ISQL_w

1. Enter your server name, login ID, and, if applicable, your password. The ISQL/W screen appears.
2. Select the applicable database from the **DB** drop-down list box.
3. Click **Load SQL Script** on the toolbar.
4. Select the MSS_BLD.SQL file from the *Ardent\DataStage\Tutorial* folder and click **Open**.
5. Click the **Execute Query** icon on the toolbar.

You may see error messages on the **ISQL/w Results** page such as: Cannot drop the table 'xxx', because it doesn't exist in the

system catalogs. These messages are harmless and are a result of the script trying to remove nonexistent tables.

Close the session to end the SQL environment.

For Oracle 8. Do one of the following:

- On Windows NT 4.0, choose **Start ► Programs ► Oracle for Windows NT ► SQL Plus *n.n***.
- On Windows 98, choose **Start ► Programs ► Oracle for Windows 98 ► SQL Plus *n.n***.

1. Enter your user name, password, and your remote connect string, if applicable.
2. Enter the following in the Oracle 8 SQL*Plus window at the SQL prompt:

```
SQL> @ora_bld.sql
```

Note: If your Ardent installation is not on drive C, substitute the correct drive letter.

When the script completes, *Finished!* appears and the SQL prompt returns.

3. End the SQL environment session. In SQL*Plus, enter **EXIT**, or simply close the window.

For Sybase. There are two ways to execute the creation script:

- On Windows NT 4.0 or Windows 98, choose **Start ► Programs ► Sybase for Windows NT ► WISQL32**.
- At the command line (DOS prompt), enter the following:

```
ISQL -e -I syb_bld.sql -o build.log -Username -Ppassword  
-Sservername
```

1. If you use the first method, enter your user name, password, database name, and server name in the WISQL32 window.
2. In the command window, issue the following statement:

```
@syb_bld.sql
```

For UniVerse. You should be familiar with UniVerse and SQL. You need sufficient access rights and SQL DBA privilege. Configure your DataStage project for use with the tutorial as follows:

1. Using Explorer, File Manager, or NT Command Prompt (DOS box), copy (do *not* move) the UNI_BLD.SQL file from the DataStage tutorial installation directory to your DataStage project directory. UNI_BLD.SQL is a script that you can execute any number of times, though once should suffice.
2. Start a Telnet session, connect to *localhost* and log in. At the Account name or path prompt, enter the fully qualified pathname to your DataStage project, then press **Return**.
3. At the UniVerse prompt, copy UNI_BLD.SQL into the UniVerse VOC file:

```
>COPY FROM &UFD& TO VOC UNI_BLD.SQL OVERWRITING
```

UNI_BLD.SQL is a script that when executed, creates and populates all necessary SQL sample tables and views used by the tutorial.
4. Enter the following to execute UNI_BLD.SQL:

```
>UNI_BLD.SQL
```
5. Enter **QUIT** to exit UniVerse.
6. Return to the DataStage Manager.

DataStage Export File

The tutorial setup program installs a DataStage export file that contains a complete sample DataStage job and meta data for use in the tutorial exercises. In the first tutorial exercise you go direct to this sample job. You can immediately look at how the job has been configured, run it, and compare the input and output files. The provision of meta data for you to use in subsequent tutorial exercises saves you having to create your own meta data and lets you begin the exercises with minimal delay.

Importing the Sample Job and Meta Data

To import the job and meta data from the export file, you need to use the DataStage Manager. It is not necessary now to understand the function of the Manager. In Chapter 6 we describe the DataStage Manager and explain how you can use it to create your own meta data. While you are setting up the tutorial, simply follow the instructions below.

To open the DataStage Manager and attach to the tutorial project:

1. Choose **Start ► Programs ► Ardent DataStage ► DataStage Manager**. The Attach to Project dialog box appears:

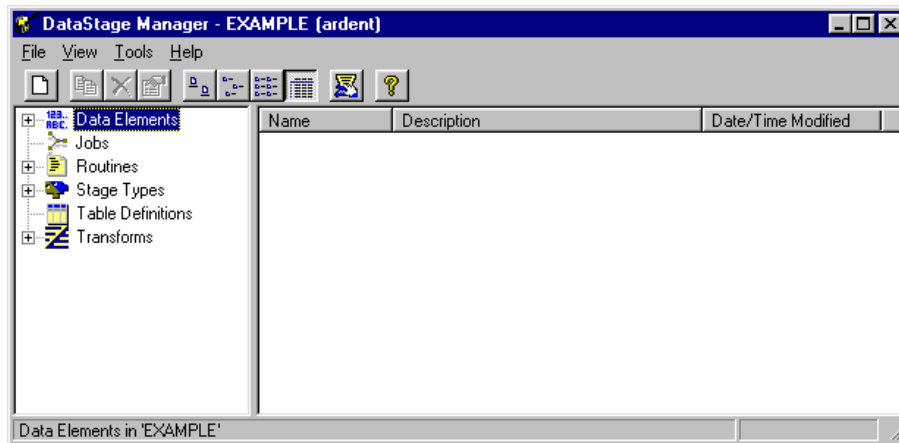
The image shows a Windows-style dialog box titled "Attach to Project". It has a blue title bar with a yellow icon on the left and a close button (X) on the right. The dialog contains several input fields and checkboxes. At the top, there is a "Host system:" label followed by a text box containing the word "ardent". To the right of this text box are three buttons: "OK", "Cancel", and "Help". Below the "Host system" section, there is a "User name:" label followed by a text box. To the right of the "User name" text box is a checkbox labeled "Omit". Below the "User name" text box is a "Password:" label followed by a text box. At the bottom of the dialog, there is a "Project:" label followed by a drop-down menu showing "EXAMPLE". To the right of the drop-down menu is a checkbox labeled "Save settings".

2. 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.
3. Enter your user name in the **User name** field. This is your user name on the server system.
4. 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.

5. Choose the project to connect to from the **Project** drop-down list box. This should be the project you selected during the tutorial setup procedure.
6. Select the **Save settings** check box to save your logon settings.

7. Click **OK**. The DataStage Manager window appears:



You can now use the DataStage Manager to import the sample job and meta data into your tutorial project:

1. Choose **Tools ► Import ► DataStage Components...**. The DataStage Repository Import dialog box appears.
2. In the **Import from file** field, use the ... (browse) button to view the destination directory you specified when you ran the tutorial setup program (by default `c:\Ardent\DataStage\Tutorial`).
3. Select the *tutorial.dsx* file.
4. Click the **Import all** option button if it is not already selected.
5. Click **OK**. The sample job, called Exercise1, its associated components, and meta data for the sample tables are loaded into the DataStage Repository.

Road Map to the Exercises

The following list describes the exercises in this manual, in the order in which you do them. Exercises 12 and 13 concern multivalued files and are specific to UniData and UniVerse users. Exercises 14 to 16 demonstrate the features of National Language Support and are specific to systems that have NLS installed. For other users, your final exercise is Exercise 11.

1. View the design and configuration of a sample DataStage job, and run the job. This exercise introduces you to the look and feel of the DataStage Designer and Director.
2. Create a job like the one you viewed in Exercise 1. It consists of Sequential File input and output stages and a Transformer stage. The source file is ITEMS.TXT. You create and populate the target file ITEMS_2.TXT, which serves as a data source in the next exercise.
3. Create a job using a Sequential File stage as the source and an ODBC or UniVerse stage as the target. You use a Transformer stage to “map” the source to the target; in this job it does not manipulate data, but simply ties the other two stages together. The exercise loads the ITEMS table in the data mart with data from the sequential file ITEMS_2.TXT, created in Exercise 2.
4. Practice what you learned in earlier exercises by creating a job on your own, using a Sequential File stage, a Transformer stage, and an ODBC (or UniVerse) stage. This exercise loads the PRICES table in the data mart with data from the PRICES.TXT sequential file.
5. Using transformations, you create the month, quarter, and year columns for the time dimension. Your only input is a sequential file, TIME.TXT, with a single date column. The exercise loads the time dimension table, TIME, in the data mart. The table consists of four columns: a date, and the corresponding month, quarter, and year. You use the QUARTER data from this file in Exercise 7.
6. This exercise demonstrates the simultaneous use of multiple sources and targets. You load the FACTS table in the data mart with the combined input of three other ODBC (or UniVerse) data sources: the ITEMS, ORDERS, and CUSTOMERS tables. You also create transformation logic to redirect output to an alternate target, the REJECTS table, using a row constraint.
7. This exercise introduces the Aggregator stage. You populate the Q_SALES summary table in the data mart using an Aggregator stage to group data from the TIME and FACTS tables. You previously populated these two tables in Exercises 5 and 6 respectively.

This completes our data mart examples. The remaining exercises demonstrate various aspects of DataStage outside the tutorial’s data mart schema described earlier in this chapter.
8. Create meta data by importing the definition of a sample database table into the Repository. This and the next exercise introduce you to the look and feel of the DataStage Manager.

9. Create table and column definitions for a comma-delimited sequential file.

This exercise takes you through the process of manually creating a definition in the Repository for the text file MYPRICES.TXT. You have the opportunity to examine the Repository and see how meta data is stored there.

10. Create a simple job that uses the meta data you created in Exercises 8 and 9.
11. In this exercise you create a job that you then step through with the job design debugger. The job populates a new sequential file, ITEMS_11.TXT, with data from the ITEMS table, which you populated in Exercise 3. You then set a breakpoint on a link in the job, and learn how to view changes in data as you step through the job.
12. This exercise and Exercise 13 are specific to users working in a UniData or UniVerse environment. In this exercise you create a multivalued hashed file, MVPRICES, from a file installed when you ran the tutorial install program. You then create table and column definitions from MVPRICES.
13. Create a job using a UniData or Hashed File stage as the source, and an ODBC or UniVerse stage as the target. The source file, MVPRICES, contains multi-valued data records, which the exercise loads in normalized form into the PRICES table in the data mart. Do this exercise only if you are a UniData or UniVerse user and you have completed Exercise 12.
14. Create a job using a Sequential File stage as the source and an ODBC, or UniVerse, stage as the target. Incoming data uses the EBCDIC character set, the job outputs the data using the ASCII character set. This demonstrates how NLS could be used, for example, to move data between mainframes using incompatible character sets.
15. This exercise demonstrates per-column mapping. Create a job using a Sequential File stage as the source and a Sequential File stage as the target. The source table has four columns which each use a different character set. The job outputs a table where all four columns use the same character set.
16. This exercise demonstrates how different locales can affect data. Create a job which you run once to sort data under the US-English locale, and again under the FR-French locale. Compare the two versions to see the different sort conventions that each locale imposes.

You may find it difficult to remember the sequence of exercises, and the way in which files and tables populated in one exercise are used as a data source in subsequent exercises. Table 2-2 provides a summary of the exercises.

Table 2-2. Summary of Tutorial Exercises

Exercise	Source	Target	Summary
Exercise1	ITEMS.TXT	ITEMS_1.TXT	Sample job. Fixed-length file to comma-delimited file.
Exercise2	ITEMS.TXT	ITEMS_2.TXT	Fixed-length file to comma-delimited file. Simple data transform of two columns.
Exercise3	ITEMS_2.TXT	ITEMS	Comma-delimited file to table. Order of input/output columns differs.
Exercise4	PRICES.TXT	PRICES	Same as Exercise3.
Exercise5	TIME.TXT	TIME	Create source meta data manually. Create three transform expressions to derive month, quarter, and year.
Exercise6	ITEMS ORDERS CUSTOMERS	FACTS REJECTS	Select source columns from three tables. Use constraints to direct output.
Exercise7	TIME FACTS	Q_SALES	Select source columns from two tables. Configure an Aggregator stage. Create a quarterly summary.
Exercise8	N/A	N/A	Import meta data from the relational database table MYPRICES.
Exercise9	N/A	N/A	Create meta data for the sequential file MYPRICES.TXT.

Table 2-2. Summary of Tutorial Exercises (Continued)

Exercise	Source	Target	Summary
Exercise10	MYPRICES.TXT	MYPRICES	A job to demonstrate the use of meta data you created in Exercise8 and Exercise9.
Exercise11	ITEMS	ITEMS_11.TXT	Simple job for debug exercise.
Exercise12	N/A	N/A	Import meta data from the multivalued hashed file MVPRICES.
Exercise13	MVPRICES	PRICES	Multivalued file normalized and output to table. Uses meta data created in Exercise12.
Exercise14	EBCDPROD.TXT	PRODUCTS	A job to demonstrate basic character set conversion features of NLS.
Exercise15	PERCOL.TXT	COLOUT.TXT	A job to demonstrate the use of per-column mapping under NLS.
Exercise16	COLLATE.TXT	COLLATE.OUT	A job to demonstrate the use of locales under NLS.

3

Sample Job

This chapter introduces you to a sample DataStage job that has already been created for you. The job, called Exercise1, was installed when you ran the tutorial install program as described in Chapter 2. It extracts data from a fixed-length sequential file and writes that data to a comma-delimited sequential file.

The job consists of the following stages:

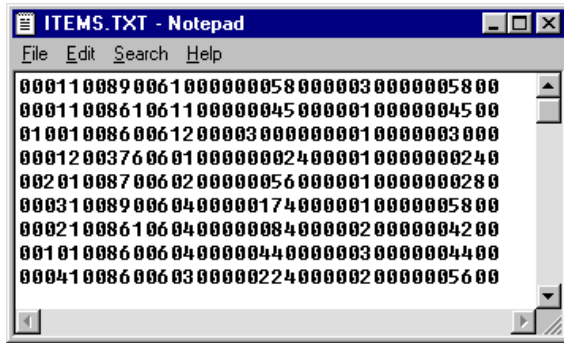
- A Sequential File stage to handle the extraction of data from the source file
- A Sequential File stage to handle the writing of data to the target file
- A Transformer stage to link the input and output columns

You look at each stage in the job and see how they are configured. You see how easy it is to build the structure of a job in the DataStage Designer and then bind specific files to that job. Finally, you compile and run the job, and then compare the source file with the target file.

This is, of course, a very basic job, but it offers a good introduction to DataStage. Using what you learn in this chapter, you will soon be creating more advanced jobs.

The source file is the fixed-length sequential file ITEMS.TXT, which was one of the sample files you installed in Chapter 2. Now is a good time to look at the file, which is located in *c:\Ardent\Datastage\Tutorial*. Open it with a text editor such as WordPad or Notepad.

It looks similar to this:



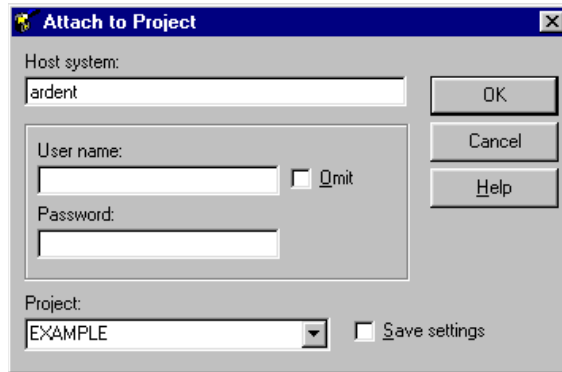
The target file is created when you run the job, and this data is written in comma-delimited format to the file.

Exercise 1: View and Run a Sample Job

DataStage jobs are opened from the DataStage Designer. You frequently use the Designer in this tutorial to create and configure new jobs and then compile them. Before you begin most of the exercises, you need to run the DataStage Designer and become acquainted with the DataStage Designer window. The tutorial describes its main features and tells you enough about the DataStage Designer to enable you to complete the exercises. For detailed information, refer to *DataStage Developer's Guide*.

Starting the DataStage Designer

Choose **Start ► Programs ► Ardent DataStage ► DataStage Designer** to run DataStage Designer. The Attach to Project dialog box appears:



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

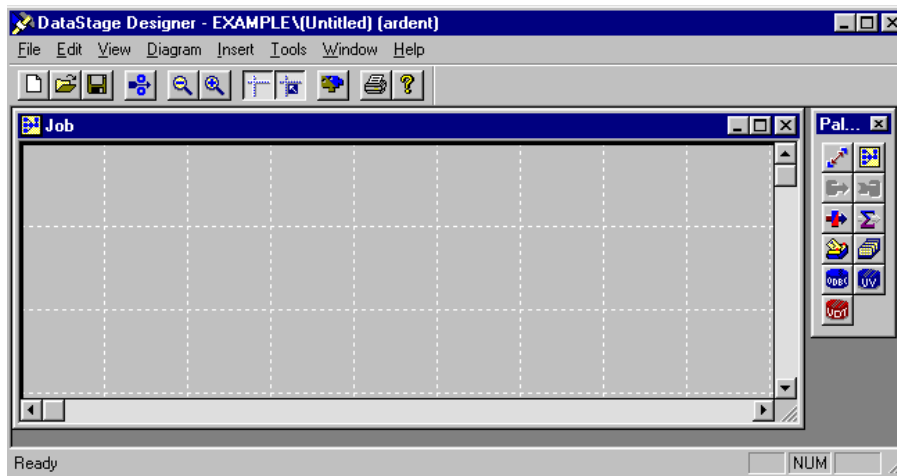
To attach to a project:

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. 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 Designer window appears:



The DataStage Designer Window

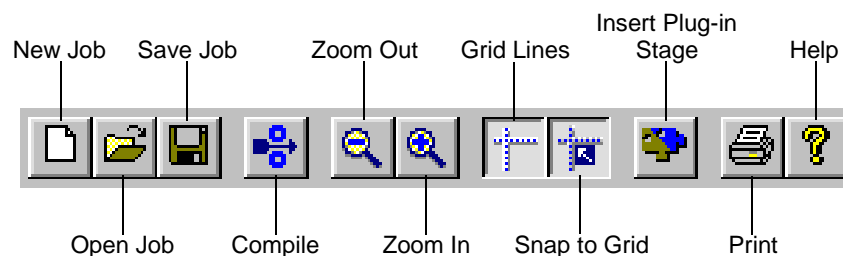
The DataStage Designer window contains a Job window, in which you design jobs, and a tool palette, from which you select job components. A status bar at the bottom of the DataStage Designer window displays one-line help for the window components and information on the current state of job operations, for example, compilation.

When you start the Designer, the Job window is empty by default. Grid lines in the window allow you to position stages precisely.

For full information about the DataStage Designer window, including the functions of the pull-down and shortcut menus, refer to *DataStage Developer's Guide*.

Toolbars

The Designer toolbar contains the following buttons:

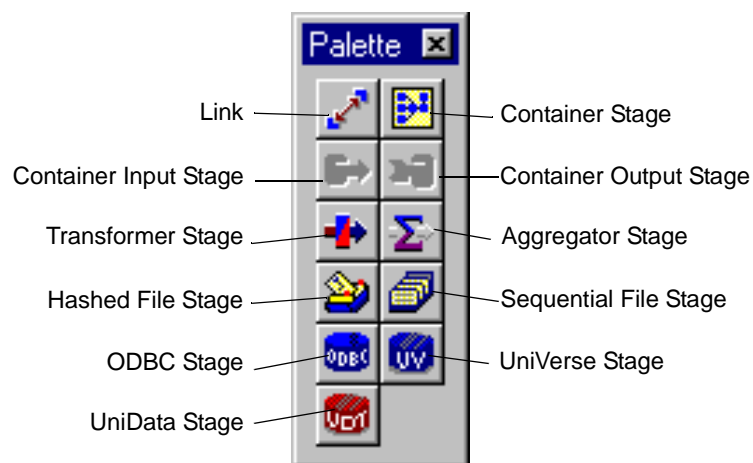


You can display ToolTips for the toolbar by letting the cursor rest on a button in the toolbar.

By default, there is also a debugger toolbar, which contains buttons representing debugger functions. Exercise 11 in Chapter 7 shows how to use the debugger and describes the toolbar. As with the Designer toolbar, you can display ToolTips by letting the cursor rest on a button in the debugger toolbar.

Tool Palette

The tool palette contains buttons that represent the components you can add to your job designs:

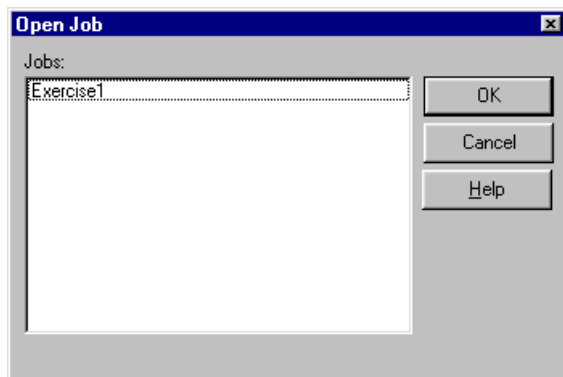


By default, the tool palette appears in the DataStage Designer window, but you can move it anywhere on the screen. To display ToolTips, let the cursor rest on a button in the tool palette.

Opening the Sample Job

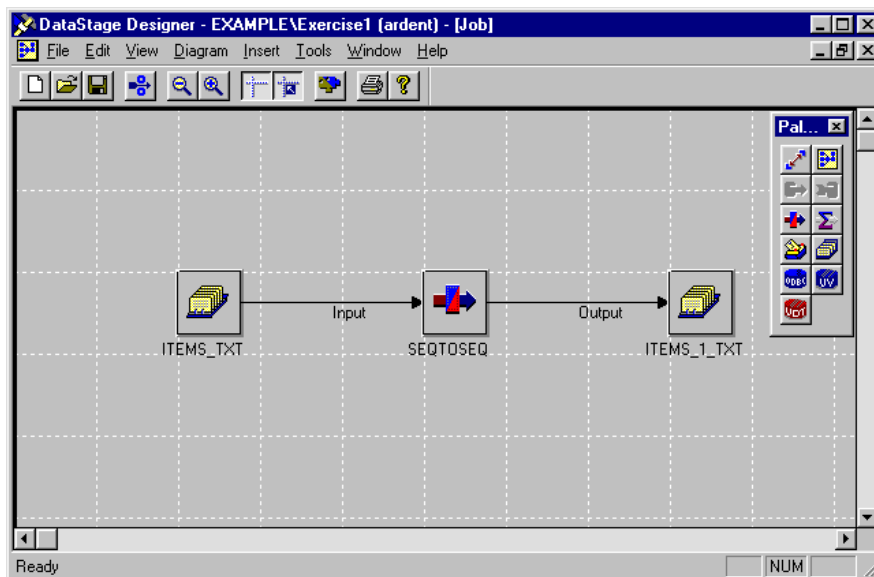
To open the sample job that we have provided for you:

1. In the DataStage Designer, choose **File ► Open Job**. The Open Job dialog box appears:



2. Double-click **Exercise1**.

The job Exercise1 appears in the DataStage Designer Job window and looks similar to this:



On the left side of the window is a Sequential File stage labelled ITEMS_TXT, and on the right side of the window is another Sequential File stage, labelled ITEMS_1_TXT.

In the center of the window is a Transformer stage, labelled SEQTOSEQ.

The stages are linked together to define the flow of data through the job. In this case, it flows from ITEMS_TXT, through the Transformer stage, to ITEMS_1_TXT.

Viewing the Configuration of the Job Stages

You can see the basic structure of the job in the Job window. The next step is to view the configuration of each stage. The configuration information identifies the specific files that are bound to each Sequential File stage, specifies the appropriate meta data, and defines what data processing is required.

Source Sequential File Stage

Let's begin by looking at the leftmost stage, ITEMS_TXT, which handles the extraction of data from a sequential file. This stage specifies:

- The location of the file from which data is extracted
 - The name of the file
 - The format of the file
 - The definition of the data columns in the file
1. Double-click the ITEMS_TXT Sequential File stage. The Sequential File Stage dialog box appears, displaying the stage **General** page. This specifies the location of the file from which data is extracted. We are using the *Tutorial* directory, into which you loaded the sample data (see "Installing Sample Data" on page 2-10), so the **Directory where files are held** field contains:

```
c:\Ardent\DataStage\Tutorial
```

The **Description** box is for the job designer to type a brief description of the stage.
 2. If you are using DataStage with NLS installed, click the **NLS** tab. The **NLS** page shows the map used for this stage. It also allows you to define a different character set map for the Sequential File stage, which overrides the default character set map set for the project or the job. However, we use the default map for this exercise.
 3. Click the **Outputs** tab. The **Output name** field at the top of the page specifies the name of the link from this stage to the Transformer stage. On the **General** page, the **File name** field contains the name of the file associated with this stage. ITEMS.TXT is the file from which data is extracted.

Note: The names given to the Sequential File stages differ slightly from the names of the associated files because stage names can contain only alphanumeric characters and underscores.

4. Click **View Data...** on the **Outputs** page. This runs the Data Browser, which displays the data in ITEMS.TXT. The names of the columns in ITEMS.TXT are displayed across the top of the Data Browser. The Data Browser is a very useful tool with which to monitor your source and target data during the tutorial exercises.

Click **Close** to close the Data Browser.

5. Click the **Format** tab. This page defines the format of ITEMS.TXT. It is a fixed-length sequential file, so the **Fixed-width columns** check box is enabled.
6. Click the **Columns** tab to display the **Columns** page. Here you see the definition of the columns in ITEMS.TXT, that is, the meta data. There is a row of meta data for each column in ITEMS.TXT. You see how to load the definitions from the DataStage Repository by using the **Load...** button later in the tutorial.
7. Click **OK** to close the Sequential File Stage dialog box. The input stage for the job is now complete.

Target Sequential File Stage

Next, let's look at the output stage for the job. It is also a Sequential File stage, and the dialog box is almost identical to that for the input stage.

1. Double-click the ITEMS_1.TXT Sequential File stage. The Sequential File Stage dialog box appears.

Notice that the dialog box for this stage shows an **Inputs** tab instead of an **Outputs** tab. This is the last stage in the job, and therefore has no outputs to other stages. It only accepts input from the previous stage.

2. The procedure for defining the target file is very similar to that for the source file.

Again, there is a field called **Directory where files are held**. It does not use the information in the input Sequential File stage, because you may want to direct the output to a different directory.

For the sake of simplicity, however, this job keeps all your files together by using the same directory as before. The **Directory where files are held** field contains:

```
c:\Ardent\DataStage\Tutorial
```

3. If you are using DataStage with NLS installed, click the **NLS** tab. The NLS map used for the stage is highlighted. We are using the default map.
4. Click the **Inputs** tab. The **Input name** field specifies the name of the link, Output. This is because the stage takes input from the link between it and the Transformer stage, rather than from the Transformer stage. On the **General** page, the **File name** field specifies that the output file is named ITEMS_1.TXT.

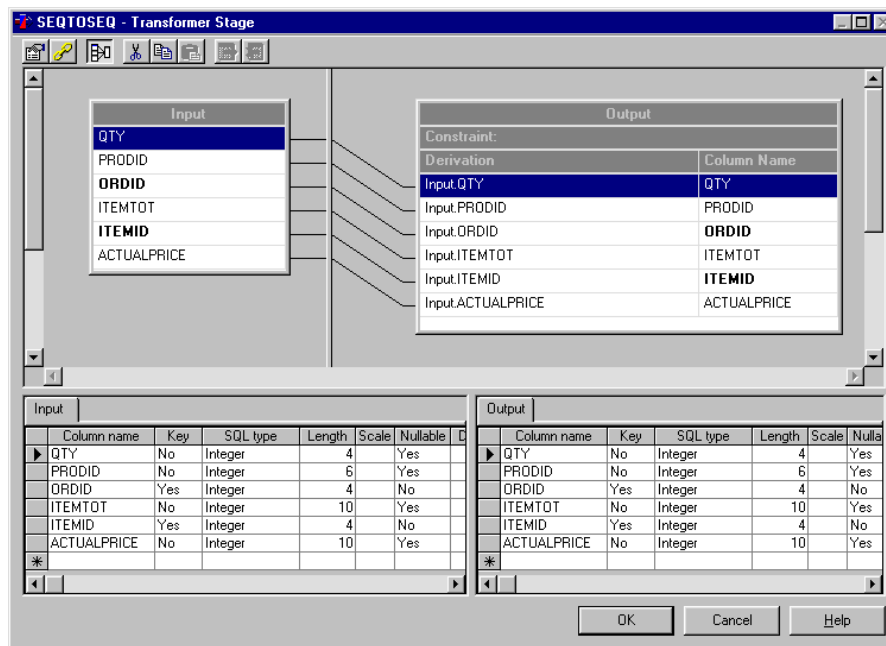
Notice that this page also has the **Overwrite existing file** radio button selected in the **Update action** group. It specifies how the output file is updated if it already exists. The first time you run the job it creates ITEMS_1.TXT, because the file does not exist; then on subsequent runs it overwrites the file.

Note: The absence of the file until you have first run the job means that an error message is displayed if you click **View Data...** now.
5. Click the **Format** tab. This page defines the format of ITEMS_1.TXT. The output data is written to a comma-delimited file, so the comma delimiter is specified and the **Fixed-width columns** check box is not selected.
6. Click the **Columns** tab and view the column definitions for the data in ITEMS_1.TXT. Exercise1 is a simple job that does not change the field formats, so the output stage uses the same column definitions as in the input stage.
7. Click **OK** to close the Sequential Stage dialog box.

Transformer Stage

The next step is to look at the Transformer stage. It connects the input and output links and specifies what transformations are to be applied to the data before it is output to the target file. In Exercise1 there are no transformations, to keep the job simple. You learn how to define transformations later in the tutorial.

1. Double-click the SEQTOSEQ Transformer stage. The Transformer Editor appears:

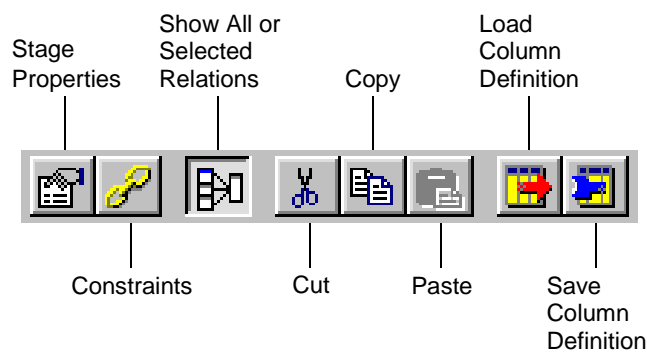


The upper part of the Transformer Editor is called the Links area. It is split into two panes, the left pane showing the columns on the input link, the right pane showing the columns on the output link. The Links area also shows the relationships (indicated by lines) between the input and output columns. In the job Exercise1, each column on the output link is derived from the equivalent column on the input link, as indicated by relationship lines between input and output columns. The **Derivation** cells on the output link are where you would specify what transformations you want to perform on the data.

Beneath the Links area is the Meta Data area. It is also split into two panes, with meta data for the input link in the left pane, that for the output link in the right pane. These panes display the column definitions you viewed earlier in the exercise on the **Columns** pages in the source and target Sequential File Stage dialog boxes.

Note: A great feature of the DataStage Designer is that you only have to define or edit something on one end of a link. The link causes the information to automatically “flow” between the stages it connects. In the job Exercise1 the developer had only to load the column definitions into the source and target stages. These definitions then appeared automatically in the Transformer Stage dialog box.

The Transformer Editor toolbar contains the following buttons:



You can view ToolTips for the toolbar by letting the cursor rest on a button in the toolbar.

For a detailed description of the Transformer Editor, refer to *DataStage Developer's Guide*. However, the steps in the tutorial exercises tell you everything you need to know about the Transformer Editor to enable you to run the exercises.

2. Click **OK** to close the Transformer Editor.

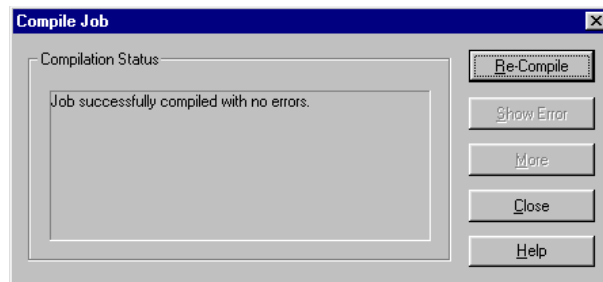
Compiling the Job

When you have designed a job and configured the individual stages, the next task is to compile the job, which you can then run. So far you have just looked at the sample job. Now you begin to use DataStage, starting by compiling the job Exercise1.

To compile the job:

1. With the job still displayed in the DataStage Designer, choose **File ► Compile** from the Designer menu or click the **Compile** button on the toolbar.

The Compile Job window appears and reports on the progress of the compilation:



Note: If you have changed anything in the job, DataStage tells you the job is modified and asks whether you want to save it. If you have made the changes accidentally, click **Cancel**, then reopen the job by choosing **File ► Open Job** from the DataStage Designer menu. When DataStage prompts you first to save job Exercise1, click **No** again.

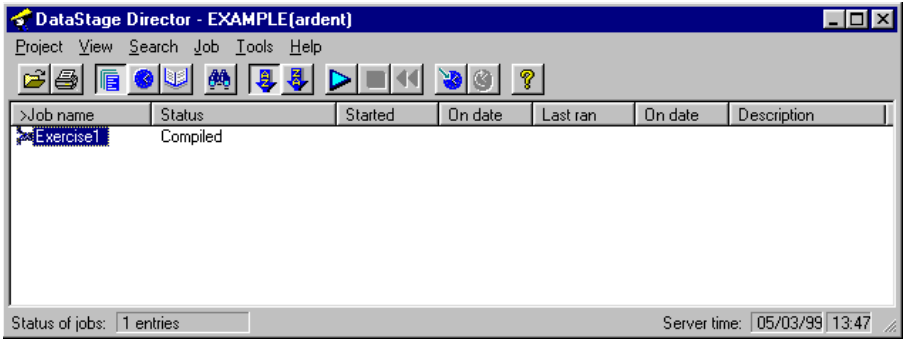
2. Close the Compile Job window when the job has been compiled.

Opening the DataStage Director

You are now ready to validate and run the compiled job, but to do so you need to start the DataStage Director. This section provides a brief introduction to the Director, then takes you through the steps to validate and run the sample job.

Choose **Tools ► Run Director** from the Designer to open the DataStage Director. You are automatically attached to the same project and you do not see the Attach to Project dialog box.

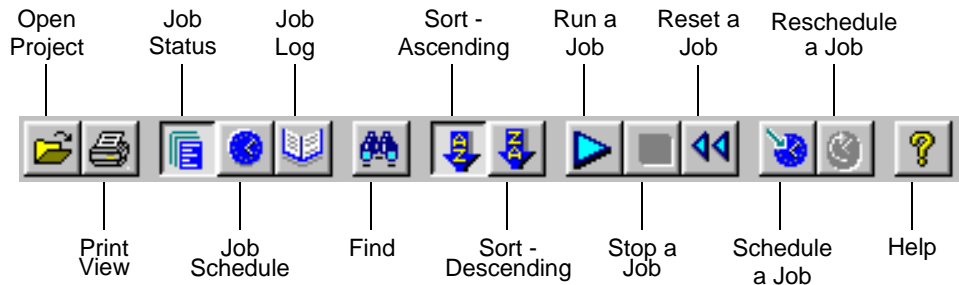
The DataStage Director window appears, by default displaying the Job Status view. You see the job Exercise1 listed with the status shown as Compiled.



The Job Status view shows the status of all jobs in the current project. You can use the View menu to switch the view to the log file for a chosen job when you run a job.

Information is taken from the server at regular intervals, or you can choose **View ► Refresh** to update the screen immediately.

The Director toolbar contains the following buttons:



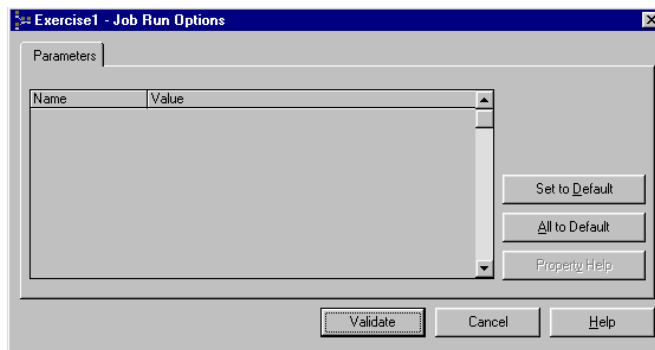
You can display ToolTips by letting the cursor rest on an icon in the toolbar.

For a detailed description of the DataStage Director, refer to *DataStage Operator's Guide*.

Validating and Running the Job

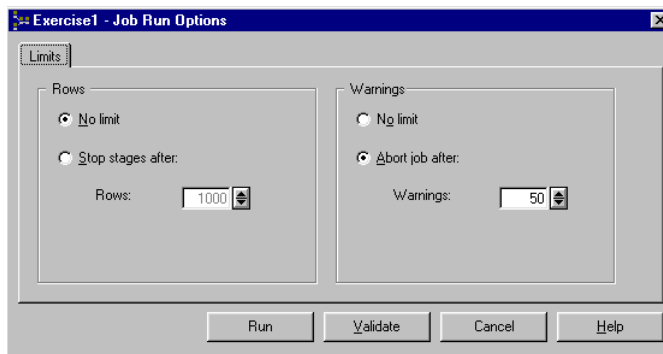
Now that you have opened the DataStage Director, continue by *validating* and running the sample job you have just compiled. In validating the job, you ask DataStage to give it one more look before finalizing it.

1. Click **Exercise1** to select it.
2. Choose **Job ► Validate**. The Job Run Options dialog box appears and lets you specify any required parameters. There are no parameters defined for this job, so click **Validate** to continue.



The status changes to Validated OK.

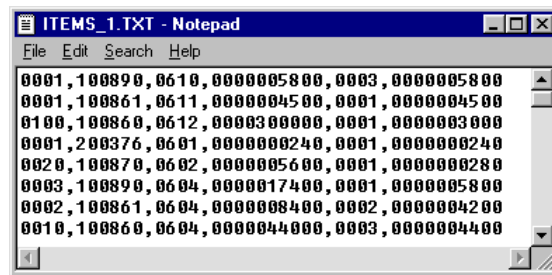
3. Choose **Job ► Run Now...**. The Job Run Options dialog box appears and collects any job run limits that are required. Click **Run** to run the job.



At this point, you can choose **View ► Log** and watch the process as it is recorded in the log file, although the job runs so quickly that it may already have finished.

The status in the job status view changes to Finished. That completes your first data transformation.

4. Open the new file, ITEMS_1.TXT in a text editor such as Notepad or WordPad. The ITEMS_1.TXT file is located in *c:\Ardent\DataStage\Tutorial* (remember, this location was specified in the configuration of the stage ITEMS_1_TXT). The file should look like this:



Notice that the file is comma-delimited, whereas the source file was fixed-length.

This exercise has laid a good foundation. In the coming exercises, you build on what you have learned and create some complex transformations. It is easy with the right tools.

Summary

In this chapter you learned what a DataStage job looks like. Using the DataStage Designer, you viewed the Sequential File Stage dialog box and the Transformer Editor, and compiled the job. Then you used the DataStage Director to validate and run the job.

In the next chapter you find out how to create your own job with the DataStage Designer. You begin by creating a job similar to the job Exercise1, except that you also transform some of the data before it is loaded into the target file.

4

Transforming Data

The four exercises in this chapter focus on extracting data from sequential files, transforming it, and loading it to a sequential file or a relational database table. You do the following:

- Create new jobs in the DataStage Designer
- Define Sequential File stages to handle your input
- Define Transformer stages to process the data
- Define the stages required for output to a sequential file or a relational database
- Save and compile the jobs
- Validate and run the jobs in the DataStage Director
- Use the Data Browser to check the results

In Exercise 2 you create a job consisting of Sequential File input and output stages and a simple Transformer stage. The job is identical to the job Exercise1, except that you also transform the data in two of the columns before it is loaded into the target file.

In Exercise 3 you populate an ODBC or UniVerse table with data from a sequential file. You are introduced to ODBC or UniVerse stages and the stage dialog box. You also use the Data Browser to view the data written to the target table.

Note: If you created sample tables in a UniVerse database in Chapter 2, you should use a UniVerse stage in Exercise 3. If you created tables in one of the other databases listed in Chapter 2, use an ODBC stage. All the exercises use ODBC rather than UniVerse stages, but the two types are very similar and both access data using SQL. You are told when there are any differences for UniVerse users.

Exercise 4 is very similar to Exercise 3, populating a table with data from a sequential file but there are no individual steps. This gives you the opportunity to find out how much you have remembered and understood from the previous exercises.

In Exercise 5 you create your data mart time dimension by extracting and transforming dates from a sequential file and loading the results into the TIME table. This exercise demonstrates the flexibility that DataStage has to transform a given source field any number of times within a single stage. It also shows how you can direct the transformed output from a single source field to several target fields.

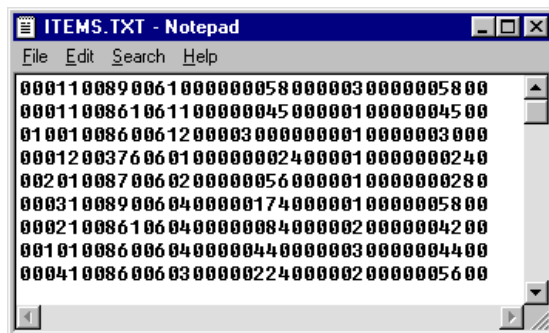
Note: For UniVerse users, the TIME table is called TIME_ because TIME is a reserved word in UniVerse.

Exercise 2: Load Data into a Sequential File

In this exercise you find out how to create a job like the sample job described in Chapter 3. You do the following:

- Create a new job in the DataStage Designer
- Configure a Sequential File stage to handle the input
- Configure a second Sequential File stage for output to a comma-delimited file
- Configure a simple Transformer stage to process the data
- Compile the job
- Validate and run the job in the DataStage Director, and check the results

You start with the fixed-length text file, ITEMS.TXT, as used in Exercise 1. It is one of the sample files you installed in Chapter 2. The file looks like this:



As with the job Exercise1, you are going to create a DataStage job that sends the output to a comma-delimited file. However, before the data is written, the job divides the data from two of the input columns, ITEMTOT and ACTUALPRICE, by 100. The effect of this is to add decimal points to the data in the ITEMTOT and ACTUALPRICE columns in the target file. At the end of the exercise you can view the new file and can contrast it with ITEMS.TXT. If this seems basic, it is. The intent is to familiarize you with creating and editing stages before you attempt more complex examples.

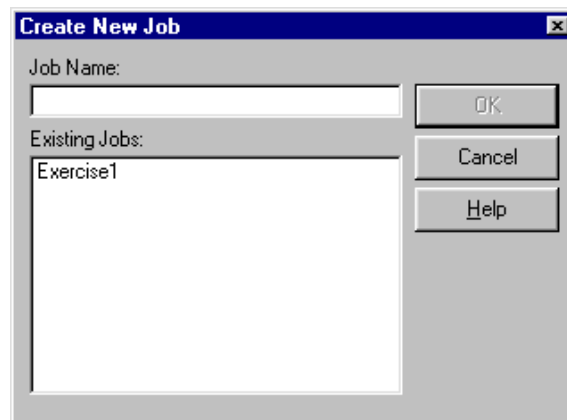
Designing the Job

Create a new job in the DataStage Designer:

1. If the DataStage Designer is still open from Exercise 1, choose **File ► New Job** to clear the Job window. An empty job is displayed.

If the DataStage Designer is not running, start it as described in “Starting the DataStage Designer” on page 3-3. An empty job is automatically displayed.

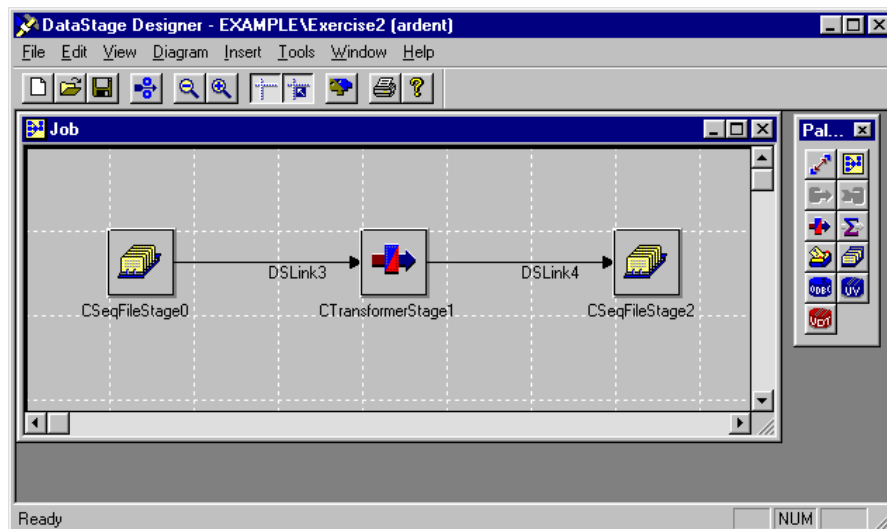
2. Save the empty job:
 - a. Choose **File ► Save Job**. The Create New Job dialog box appears:



- b. Enter **Exercise2** in the **Job Name** field.
- c. Click **OK**. The job is created and saved in the Repository.

3. Select the components for the new job from the tool palette and place them in the Job window:
 - a. Click the **Sequential File** button on the tool palette. Click in the left side of the Job window to place a Sequential File stage.
 - b. Click the **Transformer** button on the tool palette and place a Transformer stage to the right of the Sequential File stage.
 - c. Click the **Sequential File** button on the tool palette and place a Sequential File stage to the right of the Transformer stage.
4. Now link the job components together to define the flow of data in the job:
 - a. Click the **Link** button on the tool palette. Click and drag between the Sequential File stage on the left side of the Job window and the Transformer stage. Release the mouse to link the two stages.
 - b. In the same way, link the Transformer stage to the Sequential File stage on the right side of the Job window.

Your Job window should now look similar to this:



Changing Stage Names

The names of the stages and links can be altered to make it easier to identify the flow of a job, helping to make complex jobs easier to understand.

As part of this exercise, you change the names of the diagram objects, but the remaining exercises use the default object names.

Changing the name of a stage or a link is as simple as clicking it, and typing a new name or editing the old. After you edit the text, click somewhere else in the diagram, for your change to take effect.

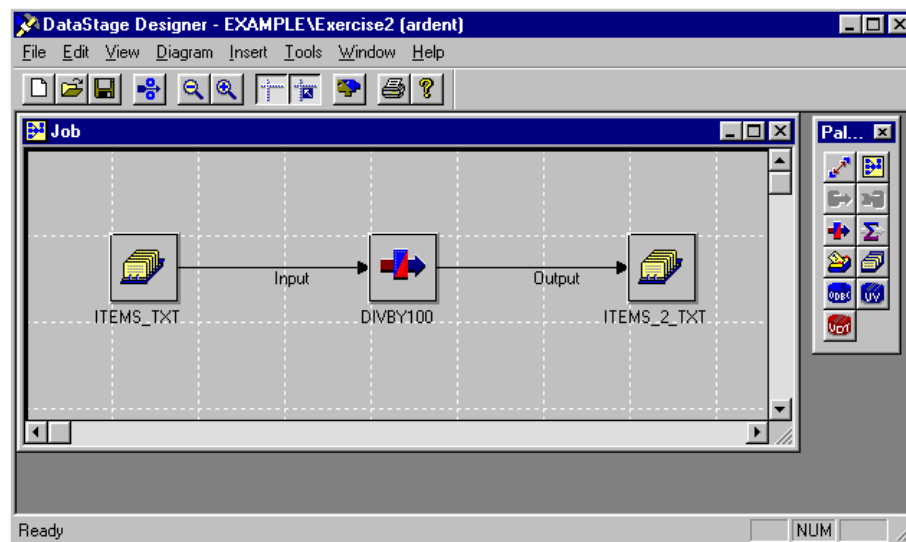
1. Click the leftmost Sequential File stage (CSeqFileStage0) and type:

ITEMS_TXT

Notice that ITEMS.TXT is not used for the name. Only alphanumeric characters and underscores are allowed. Click an empty area of the Job window to effect the change.

2. Change the name of the link between the input stage and the Transformer stage to Input.
3. Change the name of the Transformer stage to DIVBY100.
4. Change the name of the link between the Transformer stage and the output stage to Output.
5. Change the name of the output stage to ITEMS_2_TXT.

Your job window should now look like this:



Configuring the Job Stages

You have now designed the basic structure of the job. The next task is to configure each of the stages by binding them to specific files, loading the appropriate meta data, and defining what data processing you require.

Source Sequential File Stage

Let's begin with the leftmost stage, which handles the extraction of data from ITEMS.TXT.

1. Double-click the ITEMS_TXT Sequential File stage. The Sequential File Stage dialog box appears.
2. Specify the location of the file from which data is extracted. We are using the *Tutorial* directory, into which you loaded the sample data (see "Installing Sample Data" on page 2-10).

Click the **Directory where files are held** browse button (...). The Browse directories dialog box appears. Browse to the Tutorial directory (by default *c:\Ardent\DataStage\Tutorial*) and click **OK**. The Tutorial directory path is entered into the **Directory where files are held** field.

3. Specify the name of the source file (ITEMS.TXT) and its format (fixed width):
 - a. Click the **Outputs** tab. The **General** page is active by default.
 - b. In the **File name** field, type:
`ITEMS.TXT`
 - c. Click the **Format** tab to display the **Format** page.
 - d. Select the **Fixed-width columns** check box.
4. Now load the table definition for ITEMS.TXT from the DataStage Repository:
 - a. Click the **Columns** tab to display the **Columns** page.
 - b. Click the **Load...** button. The Table Definitions window appears.
 - c. Under the **Sequential** branch, there should be a folder with the same name as your project. Expand the folder and select the ITEMS table definition. Click **OK**. The table definitions appear on the **Columns** page in the grid.
5. Click **OK** to close the Sequential File Stage dialog box.

You have finished defining the input stage for the job. See how easy it is to build the structure of a job in DataStage Designer then bind specific files to that job.

Target Sequential File Stage

Next you define the output stage for the job.

1. Double-click the ITEMS_2_TXT Sequential File stage. The Sequential File Stage dialog box appears.

Notice that the dialog box for this stage does not show an **Outputs** tab, but an **Inputs** tab instead. Since this is the last stage in the job, it has no outputs to other stages. It only accepts input from the previous stage.

2. The procedure for defining the target file is very similar to that for the source file.

Again, there is a field called **Directory where files are held**. It does not use the information you put into the first Sequential File stage, because you may want to direct the output to a different directory, perhaps a mapped volume on another computer.

For the sake of simplicity, however, use the same directory as before, and keep all your files together.

Click the **Directory where files are held** browse button (...) and browse to the Tutorial directory in the Browse directories dialog box. Click **OK**.

3. Specify the name of the target file and its format:
 - a. Click the **Inputs** tab. The **General** page is displayed by default.
 - b. The output file is ITEMS_2.TXT. In the **File name** field, type:
`ITEMS_2.TXT`
 - c. Note that the default setting for the Update action is **Overwrite existing file**. Keep this setting.
 - d. Click the **Format** tab. You write your output to a comma-delimited file, so keep the default settings.
4. As for the input stage, you define the data in ITEMS_2.TXT by loading a table definition from the Repository. You are not going to make any radical changes to the field formats in this simple transformation example, so you load the same column definitions as were used in the input stage.
 - a. Click the **Columns** tab.
 - b. Click **Load...**, then select ITEMS from the Sequential branch in the Table Definitions window.
 - c. Click **OK**. The definitions load into the Columns grid.

5. Click **OK** to close the Sequential Stage dialog box. You have finished creating the output stage for the job.

Transformer Stage

With the input and output stages of the job defined, the next step is to define the Transformer stage. This is the stage in which you specify what transformations you want to apply to the data before it is output to the target file.

1. Double-click the DIVBY100 Transformer stage. The Transformer Editor appears.
2. You now need to link the input and output columns and specify what transformations you want to perform on the data. You are going to map each column on the input link to the equivalent column on the output link. You also divide the data in two of the columns by 100 before it is loaded into ITEMS_2.TXT.

Ctrl-click to select the six columns on the **Input** link, then drag them to the blank **Derivation** cells on the **Output** link. Release the mouse to add the columns to the **Derivation** cells. The derivation of the data on each output link column is the equivalent input link column, as indicated by relationship lines between input and output columns.

The top pane should now look similar to this:

Input	
QTY	
PRODDID	
ORDID	
ITEMTOT	
ITEMID	
ACTUALPRICE	

Output	
Constraint:	
Derivation	Column Name
Input.QTY	QTY
Input.PRODDID	PRODDID
Input.ORDID	ORDID
Input.ITEMTOT	ITEMTOT
Input.ITEMID	ITEMID
Input.ACTUALPRICE	ACTUALPRICE

3. Use the Expression Editor to specify that the data in the ITEMTOT and ACTUALPRICE columns is to be divided by 100.

Double-click the **Derivation** cell of the ITEMTOT output link column (it contains the expression Input.ITEMTOT). The Expression Editor opens and displays the derivation in a small window.

4. Type `/100` at the end of the derivation so that it becomes:

```
Input.ITEMTOT/100
```

This specifies that data extracted from the ITEMTOT column on the input link is divided by 100 before it is sent to the ITEMTOT column on the output link.

5. Click outside the Editor window or press **Return** to close the Expression Editor.
6. Repeat steps 3 through 5 for the ACTUALPRICE derivation.
7. Click **OK** to save the Transformer stage settings and to close the Transformer Editor.

The Transformer stage is now complete and you can compile the job.

Compiling the Job

To compile the job:

1. Choose **File ► Compile** from the menu or click the **Compile** button on the toolbar. DataStage tells you that the job is modified and asks if you want to save the job.
2. Click **OK**. The Compile Job window appears and reports on the progress of the compilation.
3. When the job has compiled, close the Compile Job window.

Validating and Running the Job

To validate and run the job:

1. Choose **Tools ► Run Director** in Designer to open DataStage Director. You are automatically attached to the same project and you do not see the Attach to Project dialog box.
2. Click Exercise2 in the Job Status view, then choose **Job ► Validate**. The Job Run Options dialog box appears and lets you specify any required parameters. There are no parameters defined for this job, so click **Validate** to continue.

The status changes to Validated OK.

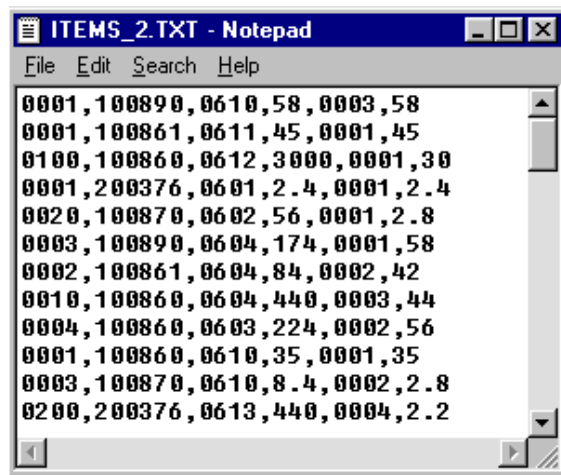
3. Choose **Job ► Run Now...** to run the job. The Job Run Options dialog box appears and collects any job run limits that are required. Click **Run** to run the job.

At this point, choose **View ► Log** to watch the process as it is recorded in the log file.

Choose **View ► Status** to return to the Job Status view.

The status in the Job Status view changes to Finished. That completes your first data transformation.

4. Now let's open the ITEMS_2.TXT file in a text editor such as Notepad or WordPad to look at the new data. The file looks like this:



Some of the data in the fourth and sixth columns now contains decimal points, thanks to the simple Transformer stage. ITEMS_2.TXT is also comma-delimited, while the source file (ITEMS.TXT) was fixed-length.

This exercise has laid a good foundation. In the coming exercises, you build on what you learned and you create some complex transformations.

We have taken you through this exercise fairly slowly because it has been the first exercise in which you created a job. In future exercises we assume that you now know enough about the DataStage Designer and Director to be able to follow briefer instructions.

Next, ODBC or UniVerse stages are introduced and you load data into a table rather than a sequential file.

Exercise 3: Load Data into a Relational Database Table

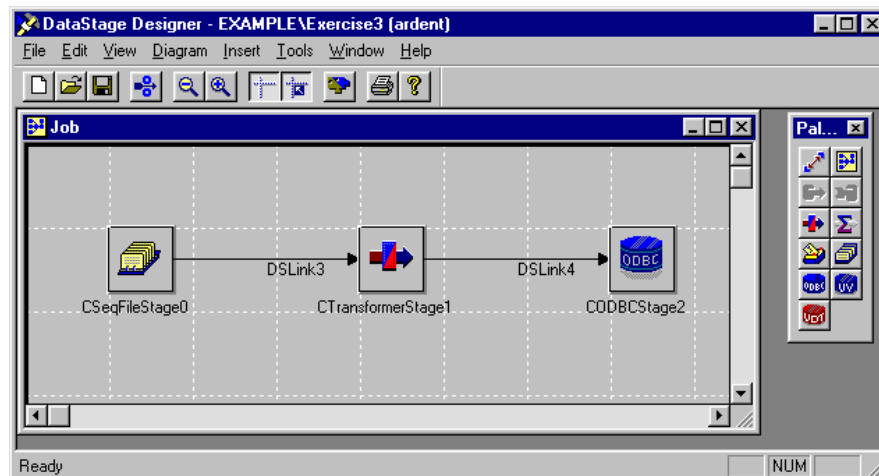
In this exercise you extract data from the sequential file you created Exercise 2, transform it, then load it to an ODBC or UniVerse target. This is a much more common scenario than the one in Exercise 2. The focus is on introducing the ODBC or UniVerse stage, so in this exercise the Transformer stage simply serves to link input and output stages, without any material transformation of the data.

Note: If you created sample UniVerse tables rather than ODBC tables when you installed the tutorial sample data (see Chapter 2), use a UniVerse stage rather than an ODBC stage.

Beginning with this exercise, the directions for steps that you performed in previous exercises are shortened. Rather than direct you to “double-click the item to display its properties” or “click **OK**”, it is assumed you are now familiar with the interface of the DataStage Designer and the Director. However, more verbose instructions are provided for new tasks.

1. Open the DataStage Designer and create a new job. Save it as Exercise3.

From left to right, add a Sequential File stage, a Transformer stage, and an ODBC (or UniVerse) stage. Link the stages together to form the job chain, as shown:



2. Edit the Sequential File stage using the Sequential File Stage dialog box. To identify the source file and load the meta data:
 - a. Click the **Directory where files are held** browse button (...) and browse to the directory for your text file (default is *c:\Ardent\DataStage\Tutorial*).
 - b. You use the output file you created in Exercise 2 as an input file for this exercise. On the **General** page on the **Outputs** page, type the file name:
`ITEMS_2.TXT`
 - c. Keep the default settings on the **Format** page, since the file is comma-delimited.
3. On the **Columns** page, click **Load...** to load the definition for the sequential file ITEMS.TXT. This meta data is close to what you need for ITEMS_2.TXT. Remember, however, that the transformation in Exercise 2 added decimal points to two of the columns in ITEMS_2.TXT, so you need to edit the column definitions to make them correct for the data in ITEMS_2.TXT.
 - a. Click the **SQL type** cell for the ITEMTOT column and choose Decimal from the drop-down list.
 - b. Click the **Length** cell for the ITEMTOT column and type 8.
 - c. Click the **Scale** cell for the ITEMTOT column and type 2.
 - d. Repeat steps a–c for the ACTUALPRICE column.

The Sequential File stage is now complete.

4. Now you set up the ODBC stage. You identify the target table, specify the table update action, and then load the meta data. Double-click the ODBC stage to display the ODBC Stage dialog box:

The screenshot shows a dialog box titled "CODBCStage2 - ODBC Stage". It has two tabs: "Stage" and "Inputs". The "Stage" tab is selected, and within it, the "General" sub-tab is active. The "General" sub-tab contains the following fields:

- Stage name: CODBCStage2
- Data source name: (empty dropdown)
- User name: (empty text box)
- Password: (empty text box)
- Transaction isolation level: None (dropdown)
- Rows per transaction: 0 (text box)
- Quote character: " (text box)
- Parameter array size: 1 (text box)
- Description: (empty text area)

At the bottom right are three buttons: OK, Cancel, and Help.

On UniVerse: If you are using a UniVerse stage, the dialog box is very similar. The only difference on this page is there is no **Parameter array size** field.

5. Select the data source name, then type your user name and password.

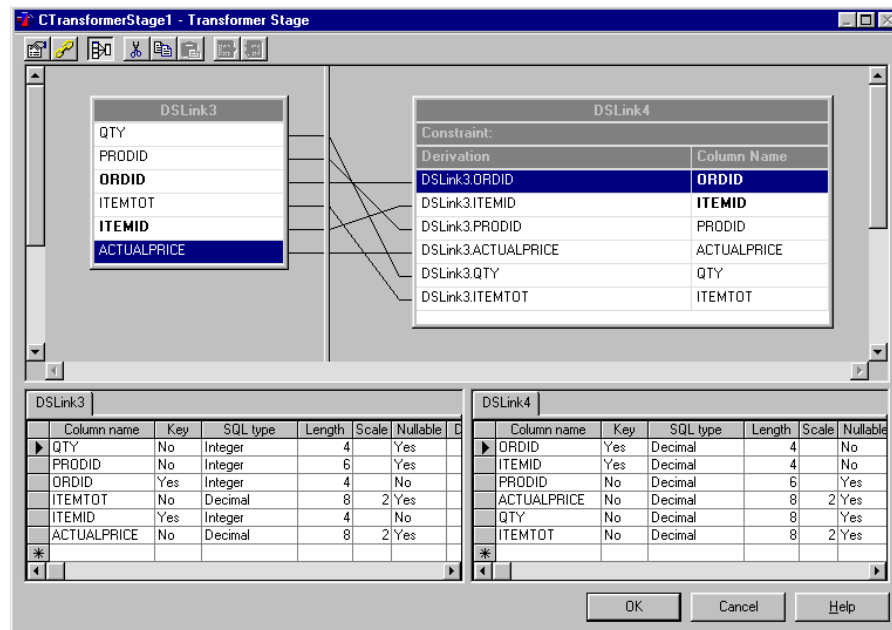
Note: Your user name and password are not required for a UniVerse stage using *localuv* as the source.

6. As in the previous exercise, you identify the data target and specify the update action. In this example the ITEMS table is the target.
 - a. Click the **Inputs** tab. The **General** page appears by default.
 - b. Select the ITEMS table from the **Table name** drop-down list box.

On UniVerse: If you are using a UniVerse stage, the tables in the **Table name** drop-down list box have your account name as a prefix, for example, EXAMPLE.ITEMS.

- c. Select **Update existing or insert new rows** from the choices in the **Update action** drop-down list box. Notice that the choices include calling a stored procedure or writing custom SQL code on the fly. (UniVerse stages do not offer the stored procedure option.) Notice that this page also has a check box **Create table in target database**, which lets you specify that the job should create a new table before writing to it.
7. Load a table definition for the table ITEMS from the DataStage Repository.
On the **Columns** page, click **Load...** to load the ODBC definition for the ITEMS table (or the UniVerse definition if using UniVerse data).
8. To see the SQL statements that DataStage uses for this job, click the **View SQL** tab.
Click **OK**. The ODBC (or UniVerse) stage is now complete.
9. Double-click the Transformer stage. The Transformer Editor appears.
Notice that as in the previous exercise the columns on the input and output links and the associated meta data cells are already filled in. You provided this information when you defined the Sequential File and ODBC stages earlier in this exercise. The only information you need to provide is for the **Derivation** column.
You are not doing any material data transformations in this exercise, and your Transformer stage serves only to join the two links. However, the order of the columns on the input and output links differs. Your Transformer stage mediates the differences by “mapping” the columns in one link to their intended counterparts in the other link.
 - a. Click the **QTY** column on the input link, DSLink3, and drag it to the blank **QTY** derivation cell on the output link, DSLink4. The expression DSLink3.QTY is added to the **Derivation** cell and a relationship line connects it to the originating input column.

- b. Repeat the process for each column on the input link, dragging it to the equivalent column on the output link. The Transformer Editor should look like this:

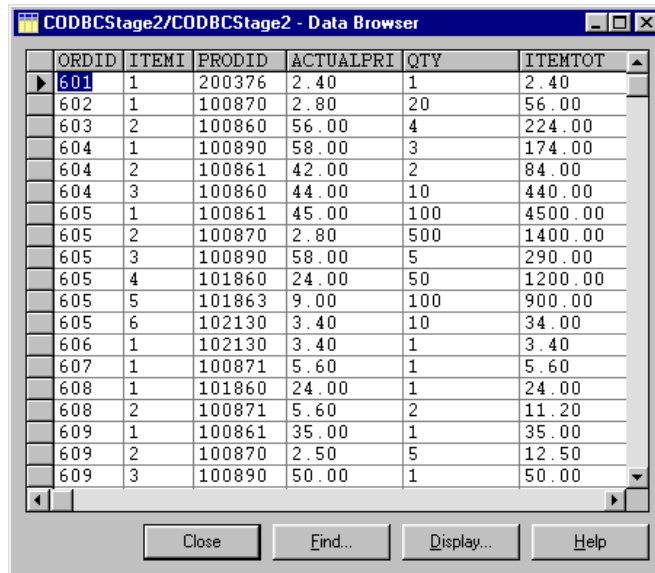


- c. Click **OK** to save the settings and exit the Transformer Editor.

The Transformer stage is now complete.

10. Save the job, then compile it.
11. Open the DataStage Director and validate the new job.
12. Run the job. The ITEMS table is loaded with data from the sequential file.
13. Now let's use the Data Browser to look at the records you have just loaded into the ITEMS table:
 - a. Return to the DataStage Designer, open the job Exercise3, then double-click the ODBC (or UniVerse) stage.
 - b. Click the **Inputs** tab, then click **View Data...**

The Data Browser displays the contents of the table:



ORCID	ITEM	PRODID	ACTUALPRI	QTY	ITEMTOT
601	1	200376	2.40	1	2.40
602	1	100870	2.80	20	56.00
603	2	100860	56.00	4	224.00
604	1	100890	58.00	3	174.00
604	2	100861	42.00	2	84.00
604	3	100860	44.00	10	440.00
605	1	100861	45.00	100	4500.00
605	2	100870	2.80	500	1400.00
605	3	100890	58.00	5	290.00
605	4	101860	24.00	50	1200.00
605	5	101863	9.00	100	900.00
605	6	102130	3.40	10	34.00
606	1	102130	3.40	1	3.40
607	1	100871	5.60	1	5.60
608	1	101860	24.00	1	24.00
608	2	100871	5.60	2	11.20
609	1	100861	35.00	1	35.00
609	2	100870	2.50	5	12.50
609	3	100890	50.00	1	50.00

If, during the exercises, you want to view the data you are extracting or the data you have loaded into a target, simply open the appropriate stage dialog box and click **View Data...** . For further information about the Data Browser, refer to *DataStage Developer's Guide*.

14. If you want to verify that *all* the records loaded from the sequential file, start your SQL environment, and issue the following statement:

```
SELECT COUNT(*) FROM ITEMS;
```

This query should yield a result of 64 rows. (If you are unsure how to start your SQL environment, see “Running an SQL Script” on page 2-13.)

Microsoft SQL Server 6.5 users: You may receive a syntax error message when issuing the above query. If you receive a syntax error message, edit the statement to remove the final semi-colon, and execute the query again.

You have successfully created a job to load data from a text file into a relational database. It is easy with the right tools.

Exercise 4: Create Your Own Job

By now you have seen how easy it is to use the tools. Everything has a logical place, and you had a chance to become acquainted with the DataStage interface.

It is easy to create a job by yourself, and you do not do anything you have not already done. Of the steps you did in Exercise 3, how many do you remember?

1. Create a job to extract data from the PRICES.TXT file, and load it into the PRICES table in the database.
2. The Transformer stage functions exactly like that in Exercise 3, that is, it serves only to map the source columns to the target columns, without any material data transformation.
3. If you run into trouble, refer to Exercise 3. The procedure for this job is virtually identical, except you don't have to edit any column definitions.

Exercise 5: Load the Time Dimension

In this exercise, you load the time dimension for the data mart. Once again, you work with a sequential file for input, and your relational database is the target. If you have been using UniVerse data up until now, continue to do so and substitute a UniVerse stage where the exercise specifies an ODBC stage.

The sequential file, TIME.TXT, contains very little raw material—just one column of dates, in the format *yyyy-mm-dd*. You build a Sequential File stage to bring in the dates, a Transformer stage to create new derivatives of the dates (month, quarter, and year), and an ODBC (or UniVerse) stage to accept the results. The results are stored in the table TIME, which has four columns containing each date and its corresponding month, quarter, and year.

Note: If you are using UniVerse, the table is called TIME_, because TIME is a reserved word in UniVerse.

You use the time dimension in Exercise 7, where it is joined with the FACTS table created in Exercise 6, to provide the source data for a quarterly sales summary.

To load the time dimension:

1. Open the DataStage Designer and create a new job. Save the job as Exercise5.
From left to right, add a Sequential File stage, a Transformer stage, and an ODBC (or UniVerse) stage. Link the stages together.

2. Edit the Sequential File stage. Click the **Directory where files are held** browse button (...) and browse to the directory for the text file (the default is *c:\Ardent\DataStage\Tutorial*).

On the **General** page on the **Outputs** page, type the file name:

TIME.TXT

Accept the default settings on the **Format** page.

3. Next you need to load a table definition for TIME.TXT. However, this meta data was not installed in the Repository when you ran the tutorial install program. The TIME.TXT file does not simulate data from an operational system like the other sequential files. Whereas we would expect to periodically load data like ITEMS and PRICES from our operational system, we only create the time dimension once. Think of it as a “disposable” data source for the purpose of this exercise.
 - a. Click the **Columns** tab on the **Outputs** page. You create a definition for the one column that TIME.TXT contains.
 - b. Click the empty row (indicated by a star) to enter column edit mode.
 - c. Type or set the column settings as follows:

Column	Setting
Column Name	DAY
Key	No
SQL type	Char
Length	10
Nullable	No
Display	10
Data element	DATE.TAG

The settings for the **Key**, **SQL type**, **Nullable**, and **Data element** cells are selected from drop-down list boxes. The SQL type **Char** matches the data element type **DATE.TAG**, since it is a string representation of a date. Leave the Scale column blank. The cells on the **Columns** page should now look like this:

	Column name	Key	SQL type	Length	Scale	Nullable	Display	Data element
►	DAY	No	Char	10		No	10	DATE.TAG
*								

4. Accept the settings. The Sequential File stage is now complete.
5. Double-click the ODBC (or UniVerse) stage to define the target table, which is **TIME**. The ODBC Stage dialog box appears. (Remember that if you are a UniVerse user, the table is called **TIME_**.)
6. Select the data source name, then type your user name and password (the name and password are not required for a UniVerse stage using *localuv* as a source).
7. Specify the target table and the update action:
 - a. Click the **Inputs** tab, then select the **TIME** table from the **Table name** drop-down list box.
 - b. Select **Insert rows without clearing** from the **Update action** drop-down list box, since the table is empty when you begin, and is only loaded once.
8. Load the ODBC definition for the **TIME** table in the database.

Click the **Columns** tab, then click **Load...** . If you are using UniVerse data, load the UniVerse definition of the **TIME_** table.

Note: If you are working in an Informix or Oracle environment, you need to edit the meta data supplied with the tutorial. Change the following settings for the **DAY** column:

On Informix: Length 13, Scale 0, Display 13

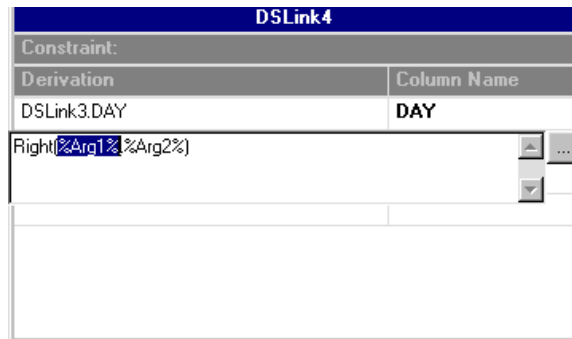
On Oracle: Length 19, Scale 0, Display 19

9. Accept the settings.
The ODBC (or UniVerse) stage is now complete.
10. Edit the Transformer stage using the Transformer Editor.
Once again, all the information is supplied except the **Derivation** field.

11. We want the DAY column on the output link to contain the date in the identical format as it is held in the source file, so drag the column DAY from the input link to the **Derivation** cell of the DAY column on the output link. A relationship line is drawn linking the two columns.
12. Next, you define the MONTH column on the output link. This is also derived from the DAY column on the input link, so drag DAY to the **Derivation** cell of the MONTH column on the output link.
13. To obtain the month from the date, you need to create an expression in the MONTH **Derivation** cell:
 - a. Double-click the MONTH **Derivation** cell to open the Expression Editor.
 - b. Click the Browse button (...) in the Expression Editor window. The Expression Editor drop-down list appears.
 - c. Select **Function...**. A list of functions appears. Select **Right** from the list of functions. It appears in the Expression Editor as shown below, with %Arg1% highlighted:

```
Right(%Arg1%, %Arg2%)DSLInk3.DAY
```

Note: The function list provides an incremental search, so if you want a function that begins with the letter R, type **R** to scroll the function list down to all the functions beginning with R. As you type additional letters, the list continues to refine the search.



- d. Type **TAG.TO.MONTH**. It automatically replaces %Arg1%.

TAG.TO.MONTH is one of an extensive set of DataStage built-in transforms (you use two others, TAG.TO.QUARTER and TAG.TO.YEAR later in the exercise). It converts a date string in the format *yyyy-mm-dd* (the format of the dates in TIME.TXT) to *yyyy-mm* format.

- e. You now extend the expression such that it results in the month being represented in the output file as a two-character string. Edit the expression so that it looks like this:

```
Right (TAG.TO.MONTH(DSLink3.DAY), 2)
```

Note: In this manual, we do not give detailed explanations of every function's arguments and syntax, because the topic is too lengthy. The exercises simply provide you with the expression examples. For further information about the functions, refer to DataStage Developer's Help.

- f. Click outside the Expression Editor window to accept the expression and close the window. The expression is added to the **Derivation** cell. Note that the cell is displayed in red if the expression contains a syntax error.

14. Using the Expression Editor, create expressions for the QUARTER and YEAR column derivations, as follows:

Column Name	Derivation
QUARTER	TAG.TO.QUARTER(DSLink3.DAY)
YEAR	TAG.TO.YEAR(DSLink3.DAY)

DSLink4	
Constraint:	
Derivation	Column Name
DSLink3.DAY	DAY
Right(TAG.TO.MONTH(DSLink3.DAY),2)	MONTH
TAG.TO.QUARTER(DSLink3.DAY)	QUARTER
TAG.TO.YEAR(DSLink3.DAY)	YEAR

When you finish, click **OK** to accept the settings for this stage and to exit the Transformer Editor.

15. Save the job, then compile it.
16. Open the DataStage Director. Your new job appears in the Job Status window with a status of Compiled.
17. Validate the job, then run it.

The time dimension is now loaded, and another step in building your data mart is complete. If you want to look at the content of the time dimension and see the effect of the built-in transforms, return to the DataStage Designer and invoke the Data Browser from the ODBC (or UniVerse) Stage dialog box in job Exercise5.

Summary

In this chapter, you learned how to create simple transformations to move data from sequential files into sequential files or database tables, using Sequential File stages, Transformer stages, and ODBC or UniVerse stages.

You used the DataStage Designer and Director to go through the process of building, saving, compiling, validating, and running jobs. In the process, you loaded three of the sample data mart tables: ITEMS, PRICES, and TIME.

You saw the differences between the various stage dialog boxes, and learned how DataStage can map a single input column to multiple output columns, all in one stage.

Next, you try some more advanced techniques. You work simultaneously with multiple sources and targets, and see how to direct output to these targets. In Exercise 7 you use the Aggregator stage to group data and create a sales summary.

5

Handling Multiple Sources

In this chapter you learn how to do the following:

- Handle multiple data sources and targets within a single job
- Use Aggregator stages to aggregate data without using SQL

In Exercise 6 you perform the crucial task of loading the fact table. Since the rows in the fact table require the primary keys of your dimension tables, you have a perfect situation to use multiple sources.

The use of multiple targets is demonstrated by constraining the loading of the fact table such that only certain rows are acceptable, while the rest are sent to an alternate table.

In Exercise 7 you use an Aggregator stage in conjunction with multiple sources.

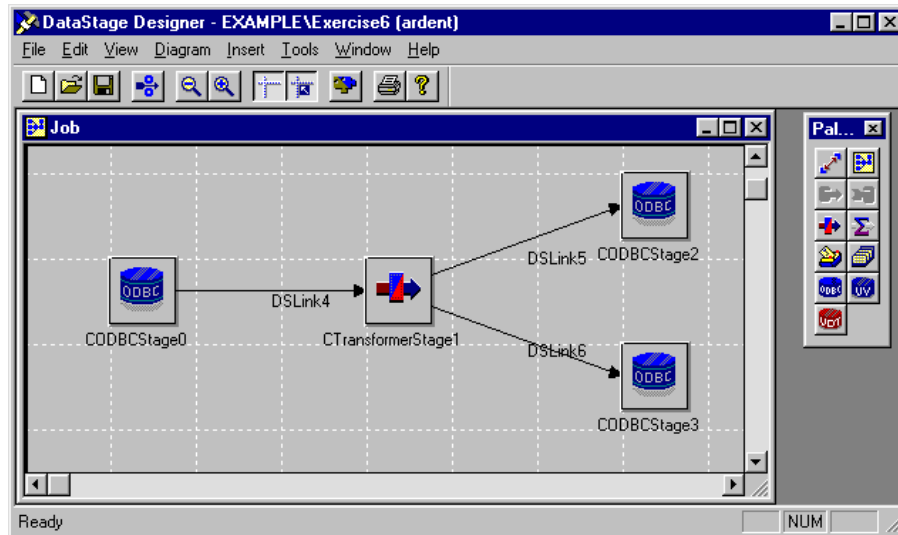
Exercise 6: Use Multiple Sources and Targets

In this exercise you create a single input stage consisting of multiple ODBC data sources, and two output stages, each consisting of a single ODBC target. Again, if you have been using UniVerse data, you continue to use UniVerse stages instead of ODBC ones.

You also create a Transformer stage, which contains some simple logic to direct the data stream to one target or the other. Most data is sent to the FACTS table, which is used as a data source in Exercise 7. The logic in the Transformer stage defines the criteria that data must meet before it is loaded into the FACTS table. Data that fails to meet those criteria is sent to the REJECTS table.

To use multiple sources and targets:

1. Open the DataStage Designer, and place four stages and three links in the Job window as shown below. If you are using UniVerse data, remember that you must use UniVerse stages rather than ODBC ones.



2. Edit the input ODBC stage, supplying your data source name, your user name, and password.
3. Specify that you want to extract data from the ITEMS, ORDERS, and CUSTOMERS tables.

On the **Outputs** page, select the ITEMS table from the **Available tables** drop-down list box. Click **Add** to add the table name to the list in the **Table names** field.

Add the ORDERS and CUSTOMERS tables in the same way.

4. Load the column definitions for each table, beginning with the ITEMS table. You are going to extract data from only some of the columns in each table, so you delete those rows in the definitions that are not required.

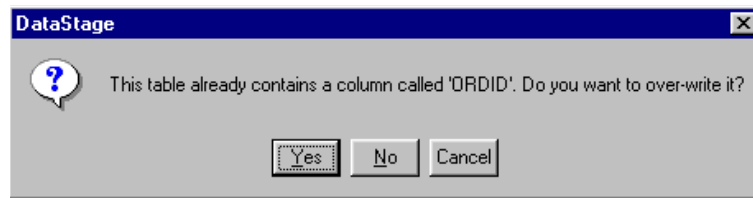
Click the **Columns** tab, then click **Load...** to load column definitions for the ITEMS table.

To delete the following rows from the definition, select the row then choose **Delete row** from the shortcut menu:

- ITEMID
- ACTUALPRICE

5. Now load column definitions for the ORDERS table.

When you attempt to load the ORDERS column definitions, the following message appears, because both the ITEMS and ORDERS tables contain a column named ORDID.



Click **Yes** to replace the ORDID definition from ITEMS with the one from ORDERS.

If you were to create a separate stage for each source, the message would not appear. But, when using multiple sources in a *single stage*, DataStage insists that the column names be unique.

You need to remove the following from the list of column definitions:

- COMMLAN
- TOTAL

Note: If you are working in an Informix or Oracle environment, you need to edit the meta data supplied with the tutorial. Change the settings for the ORDERDATE and SHIPDATE columns as follows:

On Informix: Length 13, Scale 0, Display 13

On Oracle: Length 19, Scale 0, Display 19

6. Add the definitions from the third table, CUSTOMERS.

The same warning message appears regarding the CUSTID column as for the ORDID column. Click **No** to keep the CUSTID definition from the ORDERS table.

The only column in the CUSTOMERS table that you need to keep is REPID. This column tracks which of the employees represents the customer, and is equivalent to EMPNO in the EMPLOYEES table.

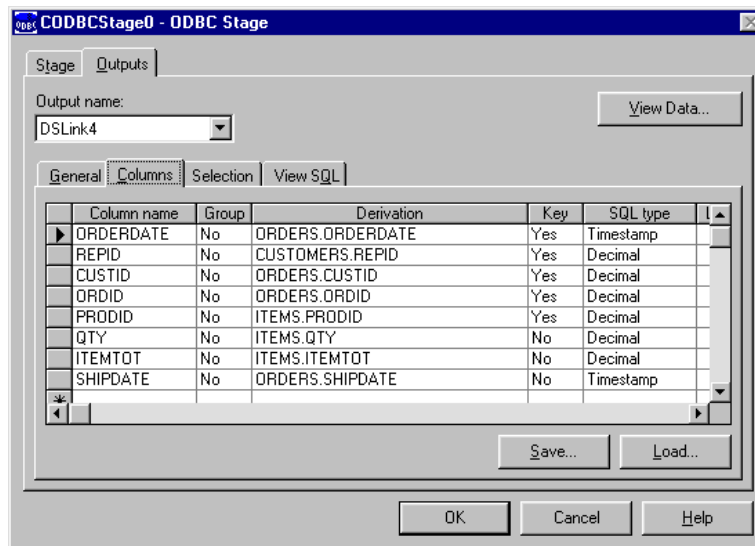
Remove the following definitions:

- NAME
- ADDRESS
- CITY
- STATE
- ZIP
- AREA
- PHONE
- CREDITLIMIT
- COMMENTS

When you configure the Transformer stage, you see that the columns you have just defined as your data source match the columns in the FACTS and REJECTS tables to which the data is to be output.

7. Now that all of the column definitions are in place, put them in the order shown below, and tell DataStage which values are keys for your source data. The new order matches the order of the columns on the target tables, which makes it easier to map the source and target columns when you edit the Transformer stage.

Edit your definitions to look like this:



Note: With UniVerse and some other RDMS products, the derivations are qualified with the name of the database account, for example, in previous examples that would be EXAMPLE. Also, Timestamp might be replaced by Date or Datetime as the SQL type in your environment.

You can change the order of the definitions by clicking the **Column name** cell and then dragging it up or down. The following illustration shows an example of the REPID column being reordered between ORDERDATE and CUSTID. Notice that the cursor has changed to a broad horizontal arrow.

When the mouse is released, the dragged column is placed *above* the row in which the cursor currently lies. In this example, releasing the mouse with the cursor in the position shown places the REPID column correctly, between the ORDERDATE and CUSTID columns. However, releasing the mouse with the cursor on the ORDERDATE/CUSTID cell border places the REPID column above the ORDERDATE column.

	Column name	Group	
	ORDERDATE	No	ORDERS.ORD
	CUSTID	No	ORDERS.CUS
	ORDID	No	ORDERS.ORD
	PRODID	No	ITEMS.PROD
	QTY	No	ITEMS.QTY
	ITEMTOT	No	ITEMS.ITEM
	SHIPDATE	No	ORDERS.SHI
	REPID	No	CUSTOMER

- Now that you have defined the three data sources, you need to join the source tables together to represent a single data stream. You use a WHERE clause to perform the table joins.

Click the **Selection** tab.

In the **WHERE clause** field, type the following:

```
ORDERS.ORDID = ITEMS.ORDID AND CUSTOMERS.CUSTID = ORDERS.CUSTID
```

Note: As previously noted, UniVerse and some other RDMS products require you to qualify these names with the account name. For example, EXAMPLE.ORDERS.ORDID = EXAMPLE.ITEMS.ORDID. The WHERE clause is case-sensitive. All parts of the clause other than the qualifying account name must always be uppercase. The case of the account name should match the name in the DataStage Designer title bar.

9. Now click the **View SQL** tab, and look at the SQL statement DataStage built from your input. Notice that the statement now includes references to all of the key values.

Accept the settings. The input stage is now complete.

10. Edit the upper-right ODBC stage to provide output to the REJECTS table.

Set the **Update action** field to **Insert rows without clearing**. In a data mart, the only time you clear rows is during a purge, when older data gets migrated to an archive.

Load the column definitions and accept the settings.

Note: If you are working in an Informix or Oracle environment, you need to edit the meta data supplied with the tutorial. Change the settings for the ORDERDATE and SHIPDATE columns as follows:

On Informix: Length 13, Scale 0, Display 13

On Oracle: Length 19, Scale 0, Display 19

11. Edit the lower-right ODBC stage to provide output to the FACTS table, using the same settings you used in step 10.

The output stages are now complete.

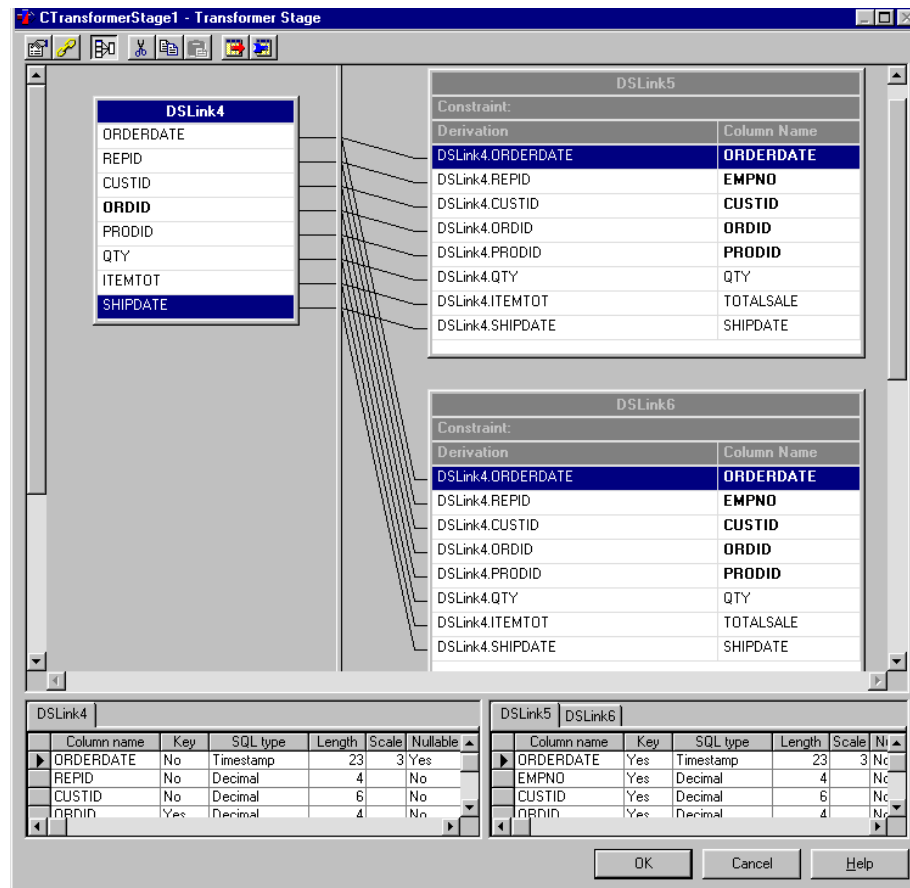
12. Add some logic to the Transformer stage to determine which records go to the REJECTS table and which to the FACTS table.

Double-click the Transformer stage to open the Transformer Editor. Notice that the Output Links area displays a separate set of columns for each output link, DSLink5 and DSLink6.

Once again, all the column information is there, except the derivations, which you define now. With two exceptions, all the derivations are direct mappings. The exceptions are as follows:

- REPID maps to EMPNO.
- ITEMTOT maps to TOTALSALE.

See the following example screen:



- Now you define a constraint expression for the FACTS link. If you have followed the exercise exactly as described, this link is DSLink6. The constraint causes the FACTS link to accept only records for which the total sale of the item is greater than or equal to \$10. Any records containing a sale for less than \$10 are rejected. (In step 14 you define the REJECTS table as the target for these rejected rows.)

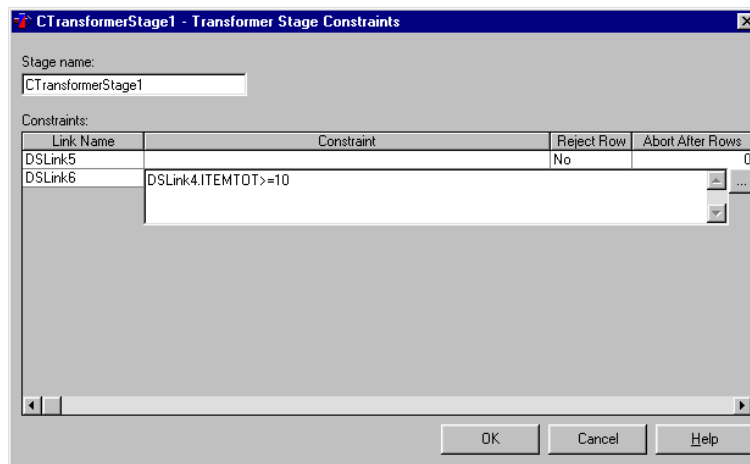
Click the **Constraints** button on the toolbar of the Transformer Editor. The Transformer Stage Constraints dialog box is displayed.

In the Constraints area, build a simple expression by double-clicking the row DSLink6 (the link associated with output to the table FACTS) in the **Constraint** field. The Expression Editor window opens.

Click the Browse button (...), then choose **Input column...** from the menu.

Select the ITEMTOT column name from the drop-down list. It appears in the Expression Editor window.

Add a “greater than or equal to 10” condition to the expression, as shown:



Click outside the Expression Editor window to close it. You have now specified that only data with a value of 10 or higher is to be written to the table FACTS.

14. Next you want to tell DataStage to use DSLink5 as the output link for any rows that do not meet the criteria defined for DSLink6.

In the Transformer Stage Constraints dialog box, set the **Reject Row** field in the row DSLink5 to Yes. This link is associated with output to the REJECTS table, so rows rejected by DSLink6 because they are for sales of less than \$10 are sent down DSLink5 to the REJECTS table.

15. Accept the settings, save the job, then compile it.
16. Open the DataStage Director, validate the job, then run it.

17. View the Job Log window in the Director. The log contains an entry indicating that four rows were rejected during the run.

These are the rows sent to the REJECTS table. Any time you designate a link to receive rejected rows, an entry is made in the log indicating how many records went through that link.

Use the Data Browser to view the data in the REJECTS and FACTS tables if you want to verify the correct operation of the job. In the Designer, double-click an output stage, select the **Inputs** page, then click **View Data...** . There should be 4 records in the REJECTS table, and all the other records (a total of 60) in the FACTS table. For further information about the Data Browser, refer to *DataStage Developer's Guide*.

Exercise 7: Aggregate Data

Suppose that senior management always wants a quarterly summary of the company's total sales volume by customer. You could get that information directly from the FACTS table, but it would be easier to have a table already built that contains the presummarized information.

Such a summary table is included in the design of the sample data mart: the Q_SALES table.

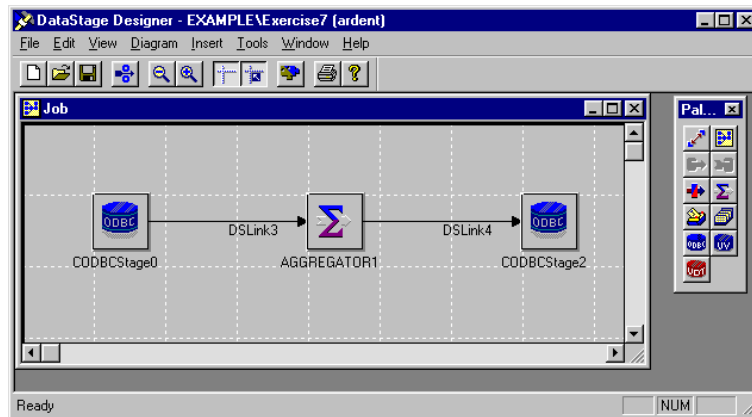
In Exercise 7, you create a job to perform the necessary data aggregation, and load the results into the Q_SALES table. The exercise introduces the Aggregator stage. It is a special kind of stage that facilitates data grouping and summarization without the need to write SQL code with lots of group functions.

You obtain data from the FACTS and TIME tables. (If you are using UniVerse the table is named TIME_.) The FACTS table provides the customer ID (CUSTID) and the value of each sale to that customer (TOTALSALE). Remember that you loaded FACTS with data from the ITEMS, ORDERS, and CUSTOMERS tables in Exercise 6. The TIME table, which you loaded in Exercise 5, supplies the quarter in which each sale occurred (QUARTER column). In a quarter there could be several sales to each customer. This exercise loads Q_SALES with the total value of the sales to each customer in each quarter.

To aggregate data:

1. In the DataStage Designer, create a new job and save it as Exercise7.

2. Add stages and links as shown:



3. Configure the input ODBC stage to use the FACTS and TIME tables as input, using only the following columns:

- QUARTER (from TIME)
- CUSTID and TOTALSALE (from FACTS)

Change the **Key** column for QUARTER to **Yes**.

Add the following WHERE clause to join the tables:

TIME.DAY = FACTS.ORDERDATE

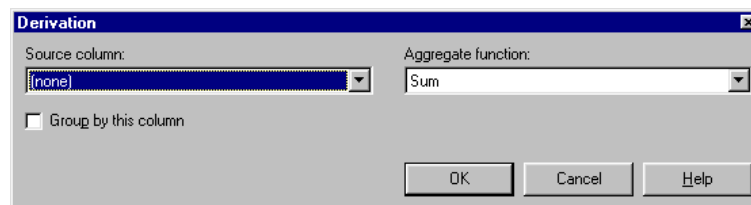
Note: As previously noted, these names may require the account name, for example, EXAMPLE.TIME.DAY = EXAMPLE.FACTS.ORDERDATE. Remember that all components of the WHERE clause are upper case, except for the account name, which must match the name in the DataStage Designer title bar. Remember also to specify the table TIME_ if you are working in a UniVerse environment.

4. Configure the output ODBC stage to use the Q_SALES table.
Select **Insert rows without clearing** from the **Update action** drop-down list box.
Load all the column definitions and accept the settings.
5. Open the Aggregator stage. The Aggregator Stage dialog box displays the Aggregator stage properties. The **Stage** page is selected by default.

Notice at the bottom of the **Stage** page that you can define before- and after-stage subroutines. Such subroutines can perform many useful functions. For example, a before-stage subroutine might check for the existence of a particular file before it allows processing to continue. An after-stage subroutine might be useful for sending an e-mail when the job finishes running.

View the **Inputs** page. It looks very much like its counterpart in other stage dialog boxes, except it contains two additional columns in the **Columns** tab: Sort and Sort Order. These columns can be used to indicate that the input data should be sorted, either in ascending or descending order. Leave them blank.

Click the **Outputs** tab. Again, the **Outputs** page appears very similar to what you have seen in other kinds of stages. Now click the **Columns** tab. The difference becomes apparent when you double-click the **Derivation** column for a given row. The Derivation dialog box appears:



The Derivation dialog box contains an **Aggregate function** drop-down list box, from which you can select various functions to aggregate data. Click **Cancel** to close the Derivation dialog box for now.

6. You want to add the value of individual sales.

Double-click the Derivation cell for the TOTALSALE row, select DSLINK3.TOTALSALE from the **Source column** drop-down list, and accept the default of **Sum** for the **Aggregate function**. Click **OK** to close the Derivation dialog box.

7. However, the sums of the sale values are to be grouped for each customer, in each quarter.

For the QUARTER and CUSTID rows, double-click the Derivation cell, set the **Source column** field to the corresponding input column, and select the **Group by this column** check box. Note that selecting the **Group by this column** check box grays out the **Aggregate function** drop-down list box.

Accept the settings.

8. Save the job, then compile it.

9. Validate the job in the DataStage Director, then run it.

You should now have 13 quarterly summary rows in the Q_SALES table. For example, in the source data there were two sales to customer 102 in 1986, quarter 2, valued at \$56 and \$224. Q_SALES now records that in this quarter the sales to customer 102 totalled \$280.

Summary

In this chapter, you learned how to create a complex transformation using multiple data sources and targets. You saw how a row of data can be “redirected” using row constraints, and accomplished the loading of the fact table in the process.

You also learned how to use an Aggregator stage to summarize and group data. Aggregator stages are extremely useful in a data warehouse setting, making the process of creating summary tables easier and more efficient.

Next, you look at how to create meta data, rather than using the table definitions installed from the tutorial CD.

6

Using Your Own Meta Data

So far in this tutorial, you have loaded predefined meta data when configuring source and target stages in your DataStage jobs. This meta data was installed when you ran the tutorial install program. We now show you how to create your own meta data for a sequential file and a relational database table.

DataStage provides two methods for creating meta data:

- The assisted approach
- The manual approach

Exercise 8 demonstrates the assisted approach. You create meta data from the table MYPRICES, importing the meta data directly into the DataStage Repository. When working with ODBC or UniVerse data sources and targets, table definitions can be quickly and easily imported into the DataStage Repository. You connect to the database and specify that you want to add MYPRICES to the Repository. DataStage simply “reads” the database definition of the table and inserts it into the Repository.

Unlike ODBC data, a sequential file contains no descriptors for the data it contains. For this reason, DataStage must rely on a “best guess” when attempting to determine what data type to use when representing each column in a file. Subtle differences between your intended data type and the data type chosen by DataStage are easily resolved with a mouse click.

In Exercise 9, you manually create the meta data for the comma-delimited sequential file MYPRICES.TXT by typing all the information into the DataStage Manager.

You use the DataStage Manager to import file and table definitions, and to create these definitions manually. Therefore, before you do Exercises 8 and 9 you need to start up the DataStage Manager, and become acquainted with the DataStage Manager window.

Note: Neither MYPRICES.TXT nor MYPRICES are part of the data mart schema outlined in Chapter 2. Although they are identical to PRICES.TXT and PRICES, which you used in Exercise 4, they are provided solely to allow you to work through the procedure for creating meta data.

The DataStage Manager

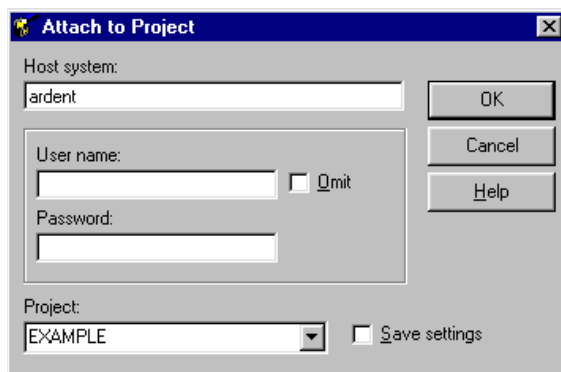
The DataStage Manager is used to:

- View and edit the contents of the Repository
- Import table definitions
- Create table definitions manually
- Create and assign data elements
- Create custom transforms and stages

For detailed information about the features of the DataStage Manager, refer to *DataStage Developer's Guide*.

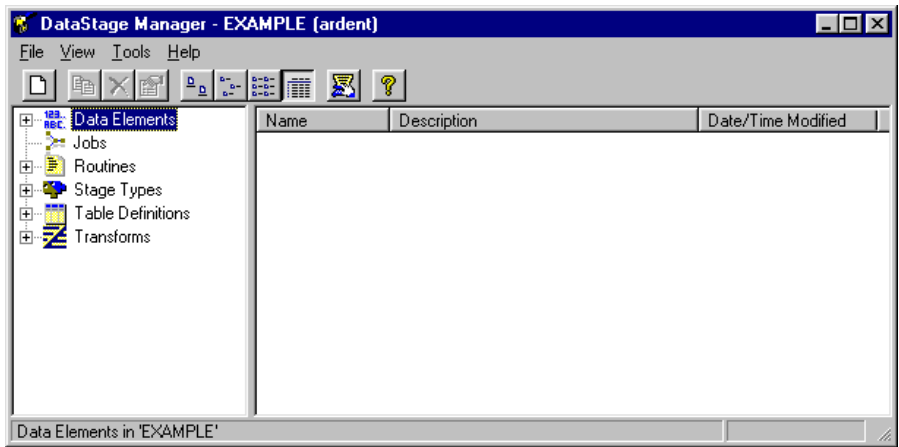
Starting the DataStage Manager

Start the DataStage Manager by choosing **Start ► Programs ► Ardent DataStage ► DataStage Manager**. The Attach to Project dialog box appears:



This dialog box also appears when you start the DataStage Designer or Director client components from the DataStage program folder. In all cases, you must attach to a project by entering your logon details. If you need to remind yourself of the procedure for attaching to the project, see page 3-3.

When you have attached to a project, the DataStage Manager window appears:

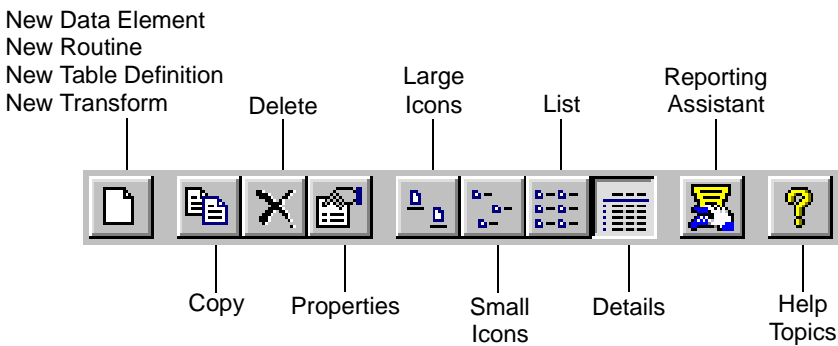


The DataStage Manager Window

The DataStage Manager window contains two panes: the left pane contains the project tree and the right pane is the display area. For full information about this window, including the functions of the pull-down menus and shortcut menus, refer to *DataStage Developer's Guide*.

Toolbar

The Manager toolbar contains the following buttons:



You can display ToolTips for the toolbar by letting the cursor rest on a button in the toolbar.

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.** This branch exists for the built-in routines and any additional custom routines you may define.
- **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 or file definition, a branch is created under the data source type (ODBC, UniVerse, UniData, Hashed, Sequential, or Stored Procedure). You see this demonstrated by the exercises in this chapter. If you manually enter a table or file 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.

Display Area

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

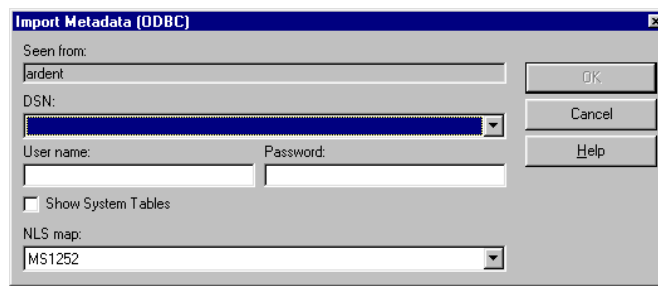
- **Large icons.** Items are displayed as large icons arranged across the display area.
- **Small icons.** Items are displayed as small icons arranged across the display area.
- **List.** Items are displayed in a list going down the display area.
- **Details.** Items are displayed in a table with **Name**, **Description**, and **Date/Time Modified** columns.

Exercise 8: Create Meta Data from a Relational Database Table

In this exercise you import a table definition (meta data) into the Repository from the sample table MYPRICES. It was created when you installed the tutorial files and ran an SQL script as described in “Creating Sample Database Tables” on page 2-13.

To import the table definition from MYPRICES:

1. From the DataStage Manager, choose **Tools ► Import ► ODBC Table Definitions...** . The Import Meta Data (ODBC) dialog box appears:



On UniVerse: Choose **Tools ► Import ► UniVerse Table Definitions...** . The Import Meta Data (UniVerse Tables) dialog box appears.

2. From the **DSN** drop-down list box, select the ODBC data source name (DSN) that corresponds to the database you used to create the tables (see Chapter 2).

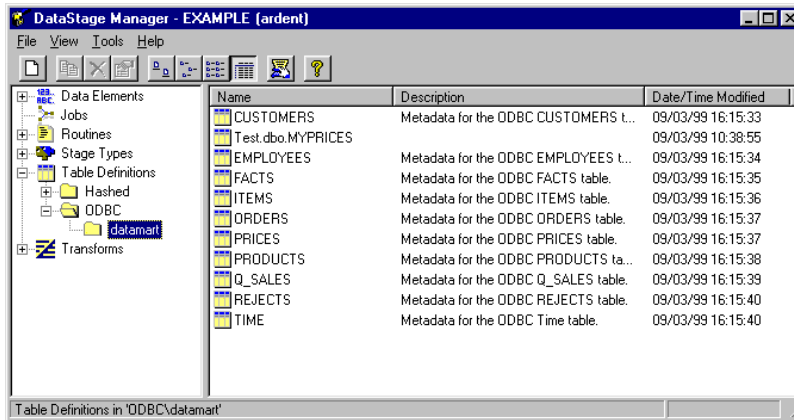
On UniVerse: From the **DSN** drop-down list box, select the *localuv* data source name (DSN) that corresponds to the database you used to create the tables. The other fields are disabled.

3. Enter the database user account name in the **User name** field. Enter the account password in the **Password** field. Click **OK**. DataStage retrieves a list of all tables in the database account.

On UniVerse: Click **OK**.

4. Select the table MYPRICES, then click **OK**. DataStage retrieves the meta data from the database, and automatically creates a table definition in the Repository.
5. Now let's have a look at the MYPRICES definition that you have just created in the Repository.

Expand the **Table Definitions ► ODBC** branch of the project tree. Your tree should resemble the following screen:



In this example, the ODBC connection used is called **datamart**, and therefore DataStage inserted a branch with that name under ODBC. This provides a handy way of keeping track of where the table definitions originated.

On UniVerse: The connection we used is called **localuv**, and therefore DataStage inserted a branch with that name under UniVerse. This provides a handy way of keeping track of where the table definitions originated. The database account name we used is called **EXAMPLE**, and DataStage prefixes all the definitions with that name.

Your table names are preceded by the database account name you used. Again, this helps to keep track of where the meta data originated.

6. Double-click the MYPRICES table to display the property sheet. It has up to four pages: **General**, **Columns**, **Format**, and, if NLS is installed, **NLS**.
7. Click the **Columns** tab. The column definitions appear.

Notice that the **Data element** column is blank. The data elements have to be entered by you. The type of data element you assign to a given column determines what transformations you can perform on that column later on. Add the data elements using the definition of PRICES in Appendix A as a guide (remember it is identical to MYPRICES).

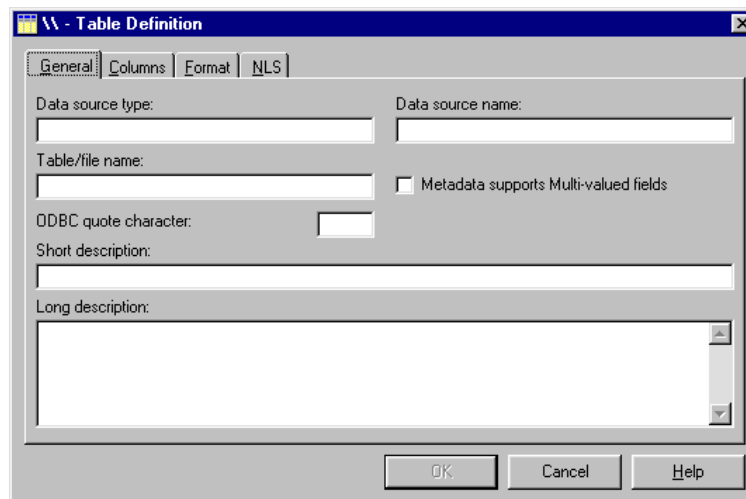
You have now defined the meta data for the table MYPRICES.

Exercise 9: Create a Table Definition Manually

To create a table definition for the MYPRICES.TXT sequential file:

1. From the DataStage Manager window, 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** button on the toolbar.

The Table Definition dialog box appears:



This dialog box has up to four pages:

- **General.** Displayed by default. Contains parameters that describe the type and name of data source and optional descriptions.
- **Columns.** Contains the column definitions for the table definition.
- **Format.** Contains file format parameters for sequential files. These fields are automatically set when you import a table definition from a sequential file.
- **NLS.** Shows the current character set map for the table definitions (this page is only visible if you are using DataStage with NLS installed).

2. On the **General** page, enter **sequential** in the **Data source type** field. The name entered here determines how the definition is stored under the **Table Definitions** branch.
3. Enter **Tutorial** in the **Data source name** field. This forms the second part of the table definition identifier and provides the name of the branch created under the data source type branch. It gives you a means to keep track of where the data definition originated.
4. Enter **MYPRICES.TXT** in the **Table/file name** field. This is the last part of the table definition identifier and provides the name of the leaf created under the data source branch.
5. Optionally, enter a brief description of the data in the **Short description** field. The text entered here is displayed in the DataStage Manager window.
6. You could enter a more detailed description of the data in the **Long description** field, which is also optional. In a real project, this information might be helpful to subsequent DataStage developers.

Entering Column Definitions

As you enter the column definitions, use the definition of MYPRICES.TXT in Appendix A as a guide, or refer to the example screen on the next page. Enter the column properties exactly as shown.

To enter the column definitions:

1. Click the **Columns** tab. The **Columns** page appears at the front of the Table Definition dialog box.
2. Click the **Column name** cell in the empty row (indicated by a star), and type **STDPRICE**, which is the name of the first column you want to add.
3. Click the **Key** cell, and choose **No** from the drop-down list to specify that this column is not a key column.
4. Click the **SQL type** cell, and choose the **Decimal SQL** data type from the drop-down list.
5. Click the **Length** cell, and type **4** to specify the data precision value.
6. Click the **Scale** cell, and type **2** to specify the data scale factor.
7. Click the **Nullable** cell, and choose **Yes** from the drop-down list to specify that the column can contain null values.

8. Click the **Display** cell, and type **4** to specify the maximum number of characters required to display the column data.
9. Click the **Data element** cell, and choose **Number** from the drop-down list to specify the type of data the column contains. This list contains all the built-in data elements supplied with DataStage and any additional data elements you have defined.
10. If you wish you can enter text to describe the column in the **Description** cell, although Appendix A does not provide sample text. 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.
11. 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 click and drag the row to a new position in the grid.

When you have finished, the table definition should look like this:

Column name	Key	SQL type	Length	Scale	Nullable	Display	Data element
STDPRICE	No	Decimal	4	2	Yes	4	Number
STARTDATE	Yes	Char	10		No	10	DATE.TAG
PRODID	Yes	Decimal	6		No	6	Number
MINPRICE	No	Decimal	4	2	Yes	4	Number
ENDDATE	No	Char	10		Yes	10	DATE.TAG
*							

12. After you have entered the column definitions, you need to specify whether the file is fixed length or delimited, as described in the following section, so do not close the Table Definition dialog box yet.

Setting the File Format

To define the file format, click the **Format** tab in the Table Definition dialog box. The left side of the **Format** page contains three check boxes, two of which are cleared by default and the third grayed out. Since this is not a fixed-width file, and the first row does not contain column names, leave them cleared.

On the right is an area for specifying a file delimiter. The default delimiter settings are those used most commonly. This file also uses the default delimiter, so do not change the setting.

Click **OK**. The DataStage Manager adds the new definitions to the Repository. MYPRICES.TXT should now be visible in the DataStage Manager display area. If it is not, expand the **Table Definitions** branch in the project tree until the MYPRICES.TXT definition is visible.

If you subsequently need to edit the definition, select MYPRICES.TXT in the DataStage Manager display area, then click the **Properties** button on the toolbar. The Table Definition dialog box reappears.

Note: You can display and edit the properties of most items in the Repository in this way.

Exercise 10: Use Your Meta Data in a Job

If you want to demonstrate to yourself that the meta data you have just created really does work, create and run a job that is like the job you created in Exercise 4. That is:

- Extract the data from MYPRICES.TXT, using a Sequential File stage.
- Pass the data through a Transformer stage with no material transformations, just a direct mapping of input to output columns.
- Load the data into MYPRICES, using an ODBC or UniVerse stage.

Since you already have the basic job design in the form of the job Exercise4, you can use that job as your starting point.

1. Open the job Exercise4 in the DataStage Designer, then choose **File ► Save Job As...** and save the job as Exercise10.
2. Edit the Sequential File stage. Change the output file name to MYPRICES.TXT. On the **Columns** page, load meta data for MYPRICES.TXT. When prompted, overwrite the meta data for PRICES.TXT.

3. Edit the ODBC or UniVerse stage. Change the input table name to MYPRICES. On the **Columns** page, load meta data for MYPRICES. When prompted, overwrite the meta data for PRICES.
4. Edit the Transformer stage. Link the input and output columns. Notice that they are not in the same order.

It is not necessary to make any other changes to the Transformer stage. Save, compile, and run the job to populate MYPRICES.

Summary

In this chapter, you learned the basics of creating meta data—your table and column definitions—in the Repository. You created meta data manually for a comma-delimited sequential file, and used the assisted approach to import meta data from an ODBC or UniVerse source.

Next you look at how to use the job design debugger to step through a job and watch changes in column data.

7

Debugging Your Jobs

The DataStage debugger enables you to test and debug your jobs. Using the debugger, you set breakpoints on the links in a job. When you run the job in debug mode, it stops whenever a breakpoint is reached, and you can then step to the next action on that link, or step to the processing of the next row of data.

Exercise 11: Use the Job Design Debugger

In Exercise 11 you do the following:

- Create a simple job on which you can run the debugger
- Run the debugger and use it to step through the design of the new job
- Compare the status of variables before and after a transformation

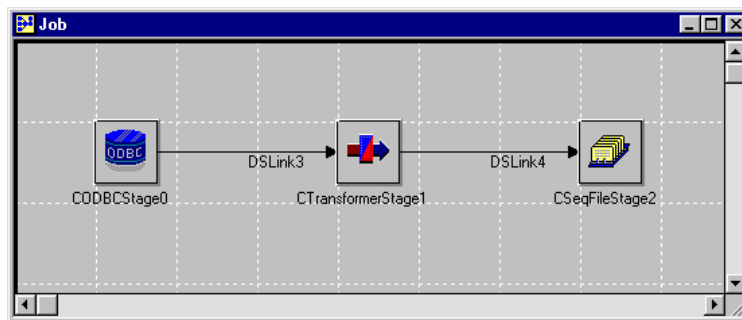
Creating a Job

In Exercise 3 you populated the ITEMS table with data from a sequential file, ITEMS_2.TXT. You now create a job in which data from the ITEMS table populates a new sequential file, ITEMS_11.TXT.

To create the job:

1. Open the DataStage Designer and create a new job. Save the job as Exercise11.

2. From left to right, add an ODBC (or UniVerse) stage, a Transformer stage, and a Sequential File stage. Link the stages together.



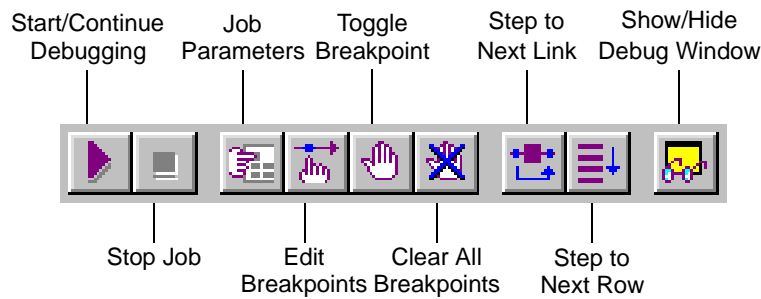
3. Edit the ODBC stage using the ODBC Stage dialog box. Begin by selecting the data source in the **Data source name** drop-down list box. On the **General** tab of the **Outputs** page, select the ITEMS table from the **Available tables** drop-down list box and click **Add** to add it to the **Table names** field. On the **Columns** page, load the definition for the ITEMS table.
4. Edit the Sequential File stage using the Sequential File Stage dialog box.
Click the **Directory where files are held** browse button (...). Browse to the directory for your text file (by default *c:\Ardent\DataStage\Tutorial*).
On the **General** page on the **Inputs** page, type the file name:
ITEMS_11.TXT
Leave the **Format** page unchanged, since the file is comma-delimited.
On the **Columns** page, click **Load...** to load the definition for the ITEMS.TXT sequential file. Remember that you added decimal points during the transformation in Exercise 2, so change the **SQL type** for the ITEMTOT and ACTUALPRICE columns to **Decimal**, with a length of **8** and a scale of **2**.
5. Edit the Transformer stage using the Transformer Editor. The columns on the input and output links have the same names, but are in a different order. Map each input column to the output column with the same name, then click **OK** to save the stage settings and exit the Transformer Editor.
6. This completes the setting up of the job you use to demonstrate the features of the debugger. Save the job and compile it.

Note: You must compile a job before you can debug it.

Running the Debugger

These steps describe how to run the debugger on the job that you have just created. It gives you an easy means of testing and debugging your job designs. For detailed information about the debugger, refer to *DataStage Developer's Guide*.

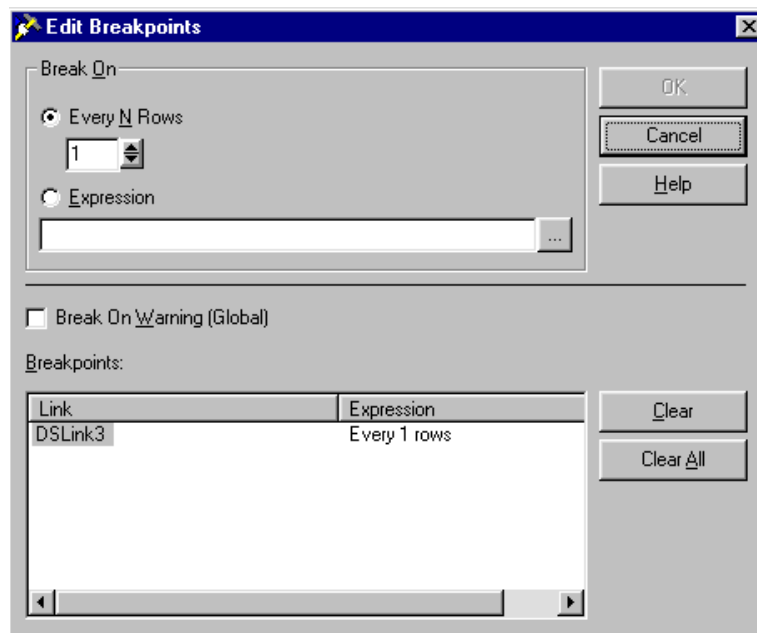
1. If the debugger toolbar is not already visible, choose **View ► Debug Bar** to display it. To display ToolTips, let the cursor rest on an button in the toolbar.



Note that you can also access these functions by choosing **File ► Debug**.

2. Set a breakpoint on DSLink3 by selecting the link, then clicking the **Toggle Breakpoint** button on the toolbar. A large black dot appears on the link where it connects to the ODBC stage; this indicates a breakpoint is set on the link.

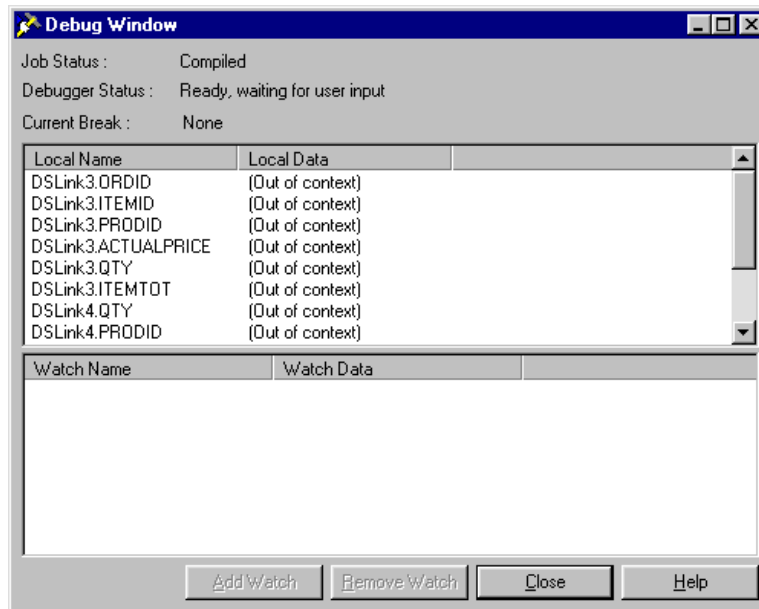
3. With the link still selected, click the **Edit Breakpoints** button on the debugger toolbar. The Edit Breakpoints dialog box appears:



You use this dialog box to specify the point at which a break occurs.

4. Click **Cancel** to accept the default setting of 1 for **Every N Rows**. This causes the job to stop after each row is processed on the link. (If you were to select an expression, a break would occur if the expression evaluated to TRUE for the specified link.)

5. Click the **Show/Hide Debug Window** button on the debugger toolbar to display the Debug Window. This window shows the current status of the job and the debugger.

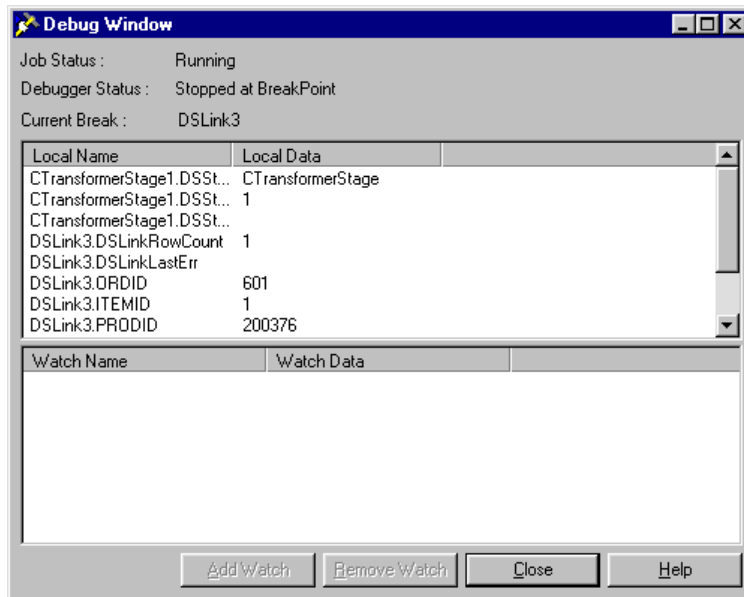


Keep the Debug Window open while you continue with the next steps.

6. Click the **Start/Continue Debugging** button on the debugger toolbar.
7. The Job Run Options dialog box appears. Click **OK** in the dialog box to run the job.

While the job is running, notice the status changes in the Debug Window. When a breakpoint is reached, the job status remains as "Running" and the debugger status changes from "Running Job" to "Stopped at BreakPoint". The current break is shown as "DSLink3".

The top pane of the Debug Window displays the names and values of variables on the link (DSLink3) and the Transformer stage (DSTransformer1). The bottom pane, which displays the watch list, is empty because you have not yet defined what you want to watch.



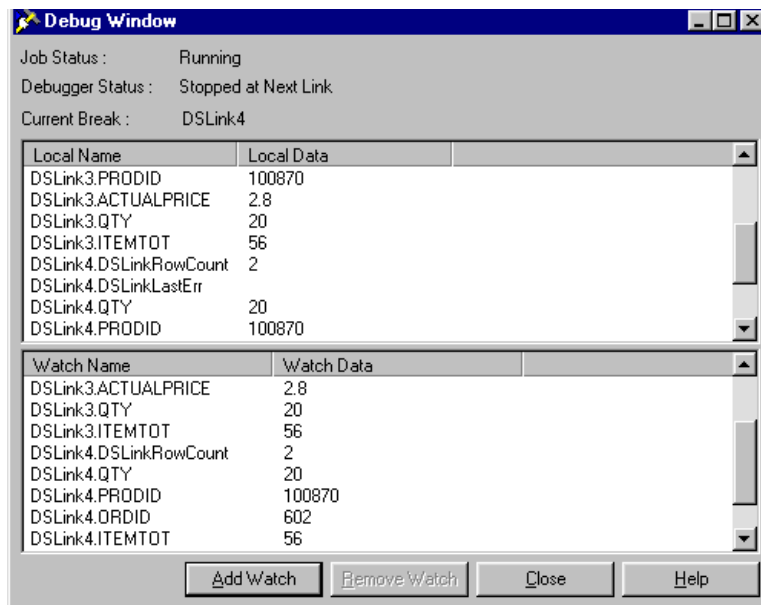
8. To add DSLink3.DSLinkRowCount to the watch list, select **DSLink3.DSLink-RowCount** from the list of variables in the upper pane, then click **Add Watch**.
9. Select each input column from the upper pane of the Debug Window (ORDID, ITEMID, and so on), then click **Add Watch** to add them to the watch list.
10. This and the following steps show you how to step through the job as it processes data.

Click the **Step to Next Row** button on the debugger toolbar to step the debugger forward to the next row processed on link DSLink3.

In the Debug Window, the status of the debugger changes to “Running” when you first click **Step to Next Row**, then becomes “Stopped at Next Row”. The watch list displays the data values for the next row of input data from the ODBC stage. For example, the DSLinkRowCount is now 2.

11. Click the **Step to Next Link** button on the debugger toolbar to step to the next link in the job design. The debugger stops at the next link, DSLink4, and the status information in the Debug Window changes accordingly. The upper pane now lists the variables on both DSLink3 and DSLink4.
12. Select **DSLink4.DSLinkRowCount** and each of the DSLink4 input columns, and add them to the watch list.

Notice that in this exercise the values on the DSLink3 and DSLink4 columns are the same, because you have not transformed the data. (You have only changed the order of the variables.) This procedure is very useful for watching column data before and after a Transformer stage to see the effects of the applied transform.



13. Click the **Step to Next Row** button. You return to the break at DSLink3. Notice that in the watch list all the DSLink4 variables are listed as being out of context. This is because in the context of DSLink3 there is no information for the DSLink4 variables.
14. Close the Debug Window.

15. Remove the breakpoint on DSLink3 by selecting the link, then clicking the **Toggle Breakpoint** button on the debugger toolbar. You can now run the job to completion by clicking the **Start/Continue Debugging** button.
16. The Debug Window appears. When the job finishes running, the Debug Window shows the current break as None, and in the watch list all link variables are out of context. Close the window.
17. As in the last part of Exercise 11, you can use the Data Browser to view the data in the file ITEMS_11.TXT.

In the Designer, double-click the Sequential File stage, select the **Inputs** page, then click **View Data...**

The Data Browser displays the data you have just written to ITEMS_11.TXT. For further information about the Data Browser, refer to *DataStage Developer's Guide*.

QTY	PRODID	ORCID	ITEM TOT	ITEM	ACTUALPRICE
1	200376	601	2.4	1	2.4
20	100870	602	56	1	2.8
4	100860	603	224	2	56
3	100890	604	174	1	58
2	100861	604	84	2	42
10	100860	604	440	3	44
100	100861	605	4500	1	45
500	100870	605	1400	2	2.8
5	100890	605	290	3	58
50	101860	605	1200	4	24
100	101863	605	900	5	9
10	102130	605	34	6	3.4
1	102130	606	3.4	1	3.4
1	100871	607	5.6	1	5.6
1	101860	608	24	1	24

Summary

In this chapter, you created a simple job to move data from a database table into a sequential file. You then learned how to run the job design debugger on this job. You set a breakpoint and saw how to view changes in data as you stepped forward to the next row or link. After you ran the job to completion, you used the Data Browser to view the data in the sequential file.

8

Working with Multivalued Files

The exercises in this chapter demonstrate the ability of DataStage to handle UniData or UniVerse databases containing *nonfirst-normal-form* (NF²) data. You do the following:

- Create a sample multivalued file (that is, a file in NF² format) and import the definition of that file into the DataStage Repository (Exercise 12)
- Extract data from the multivalued file and load it into the PRICES table in your data mart (Exercise 13)

The exercises are targeted specifically at UniData and UniVerse users. If you are a UniData user, note also that you have to use a relational database with ODBC as the target. If you do not fall into these categories of user, leave this chapter and go to Chapter 9.

Exercise 12: Import the Multivalued File Definition

In this exercise you create meta data from a multivalued UniData hashed file or a multivalued UniVerse hashed file. The focus is on using the UniData stage, but the procedure when using a UniVerse hashed file is very similar, and any differences are explained in the example.

Remember that you must be a UniData or UniVerse user to do Exercises 12 and 13.

Creating the Sample Multivalued Source File

The following procedure describes how to create and populate the multivalued source file, from which you then derive the meta data:

1. If you are working in a UniVerse environment, check whether your DataStage project directory includes a directory named BP. (In a UniData environment, this directory is already in the correct location.) If BP does not exist, create it as follows:

- a. Start a Telnet session, connect to *localhost*, and log in. At the `Account name or path` prompt, enter the fully qualified pathname to your DataStage project, then press **Return**.

- b. At the UniVerse prompt, type:

```
>CREATE.FILE BP 1
```

Keep the Telnet session open; you need it for later steps.

2. In Chapter 2, you copied sample tutorial files by running *install.exe* from the tutorial CD. Two of the files copied to the *Ardent\DataStage\Tutorial* directory were *LOADFILE.B* and *MVPRICES.TXT*. Using Explorer, File Manager, or NT Command Prompt (DOS box), copy these two files from the tutorial directory to the BP directory of the account where you wish to create the file. (On UniVerse, this is the BP directory that you have just created.)
3. If you have not already done so, start a Telnet session and log in.
4. At the UniData prompt (:) or UniVerse prompt (>), compile *LOADFILE.B*:

```
:BASIC BP LOADFILE.B
```

5. Run *LOADFILE.B* to create and populate the source file:

```
:RUN BP LOADFILE.B
```

6. You are prompted for a file name, which is the name of the text file that contains the source data. Type **BP/MVPRICES.TXT**.

A hashed file named *MVPRICES* is created in the account.

7. Enter the following command to see the content of *MVPRICES*:

```
:LIST MVPRICES
```

8. Quit the Telnet session.

Creating the Meta Data

Follow these steps to create meta data from the hashed file MVPRICES, which you created in the previous section. You use the meta data when you do Exercise 13.

1. From the DataStage Manager, choose **Tools ► Import ► UniData File Definitions...**. The Import Meta Data (Unidata Files) dialog box appears.

On UniVerse: Choose **Tools ► Import ► UniVerse File Definitions...**. The Import Meta Data (UniVerse Files) dialog box appears.

2. Enter the node name of the server, and the user account name and password used to connect to the server.

Click the **Database** drop-down list box. DataStage populates it with a list of the databases available on the server. Select the database on which the file MVPRICES is held, then click **OK**.

On UniVerse: From the **Account** drop-down list box, select the account that contains the hashed file MVPRICES.

3. The DataStage Manager searches the specified UniData database or UniVerse account, and lists the available files in the Import Meta Data dialog box. Select the file MVPRICES.
4. Click **Details...** in the Import Meta Data dialog box to view the column headings, and confirm that this is a multivalued file. The Details of message box appears. “(M)” indicates that a field contains multivalued data.



5. Close the Details of message box, then with MVPRICES still selected, click **OK**.

DataStage retrieves the meta data from the database, automatically creates a file definition in the Repository, and adds a **UniData** entry (if one does not already exist) to the table definitions in the DataStage Manager window.

On UniVerse: **Hashed**, rather than **UniData**, is added to the list of table definitions in the DataStage Manager window.

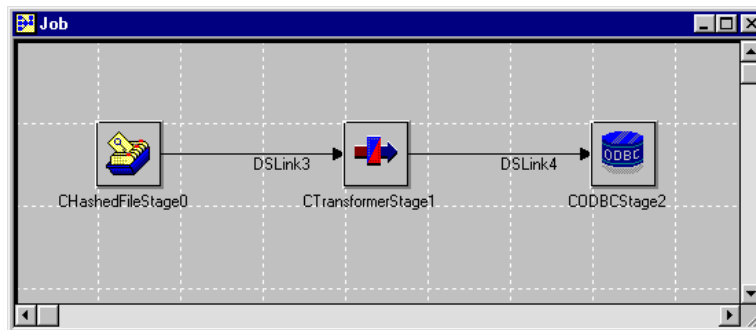
You have now created meta data in the DataStage Repository for the multivalued file. In the next exercise you extract the data from this file and load it into your data mart.

Exercise 13: Extract Data from a Multivalued File

This exercise introduces you to the UniData stage or, for UniVerse users, the Hashed File stage. You extract data from the sample file created in Exercise 12, pass the data through a simple Transformer stage, then load it in 1NF form into an ODBC or UniVerse target (in this case the PRICES table). You see the UniData Stage or Hashed File Stage dialog box, and find out how to view the data in normalized and unnormalized form.

1. Open the DataStage Designer and create a new job. Save it as Exercise13.

From left to right, add a UniData or Hashed File stage, a Transformer stage, and an ODBC or UniVerse stage. Link the stages together to form the job chain, as shown below:

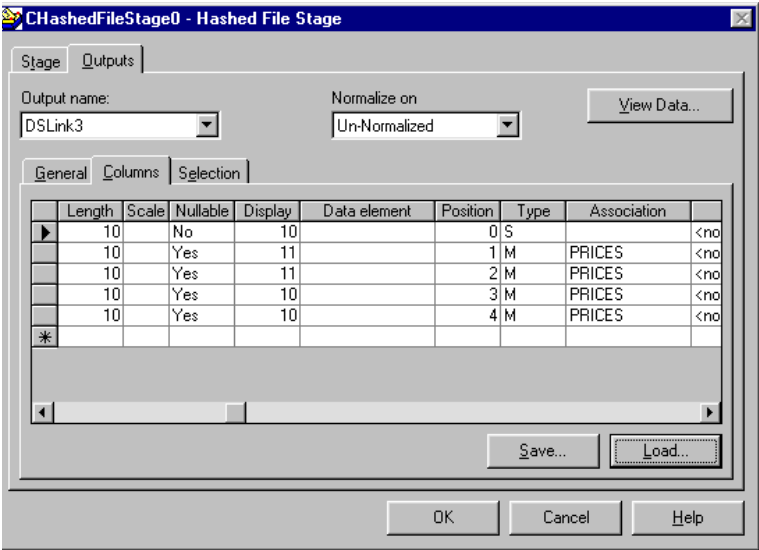


2. Edit the UniData stage, using the UniData Stage dialog box. Identify the location of the UniData file by selecting the server node name from the **Server** drop-down list box, and the database from the **Database** drop-down list box, then enter your user name and password.

On UniVerse: Edit the Hashed File stage, using the Hashed File Stage dialog box. From the **Account name** drop-down list box, select the account in which you created the hashed file.

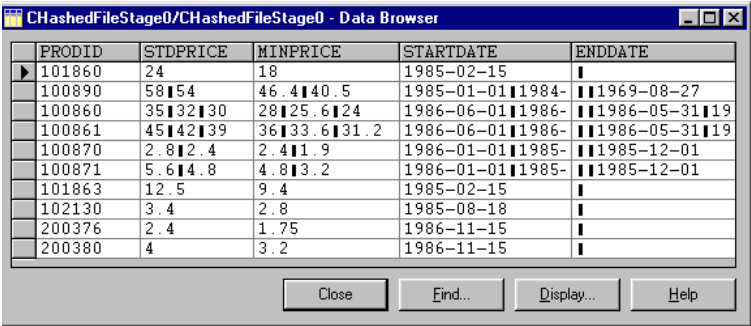
3. Click the **Outputs** tab, and on the **General** page select the MVPRICES file from the **File name** drop-down list box.

On the **Columns** page, click **Load...** to load the UniData or Hashed File definition for the MVPRICES file.



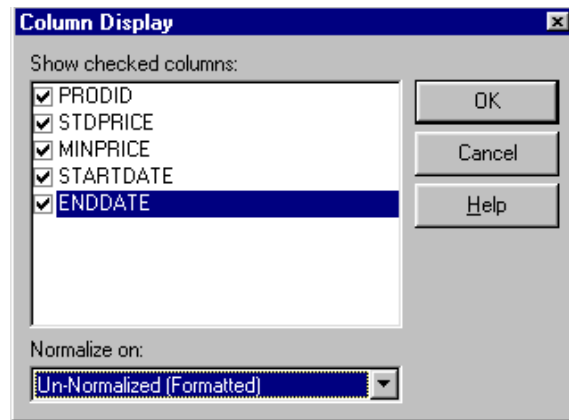
In this example screen, the **Columns** page is scrolled to the right to show the field types and associations.

4. Have a look at the data in unnormalized form by clicking **View Data...** . The Data Browser appears:



Alternatively, you might like to see a formatted unnormalized view.

- a. Click **Display...** . The Column Display dialog box appears:



- b. Select **Un-Normalized (Formatted)** from the **Normalize on** drop-down list box, then click **OK**. The Data Browser displays the multivalued rows split over several lines.
5. You now need to select the association on which to normalize the data.
Close the Data Browser and select **PRICES** from the **Normalize on** drop-down list box. To view the format of the data normalized on **PRICES**, click **View Data...** .

CHashedFileStage0/CHashedFileStage0 - Data Browser

	PRODID	STDPRICE	MINPRICE	STARTDATE	ENDDATE
▶	101860	24	18	1985-02-1	
	100890	58	46.4	1985-01-0	
	100890	54	40.5	1984-06-0	1969-08-27
	100860	35	28	1986-06-0	
	100860	32	25.6	1986-01-0	1986-05-31
	100860	30	24	1985-01-0	1985-12-31
	100861	45	36	1986-06-0	
	100861	42	33.6	1986-01-0	1986-05-31
	100861	39	31.2	1985-01-0	1986-12-31
	100870	2.8	2.4	1986-01-0	
	100870	2.4	1.9	1985-01-0	1985-12-01
	100871	5.6	4.8	1986-01-0	
	100871	4.8	3.2	1985-01-0	1985-12-01
	101863	12.5	9.4	1985-02-1	
	102130	3.4	2.8	1985-08-1	
	200376	2.4	1.75	1986-11-1	
	200380	4	3.2	1986-11-1	

Close Find... Display... Help

Close the Data Browser, then click **OK** to exit the UniData Stage or Hashed File Stage dialog box.

6. Edit the ODBC or UniVerse stage using the ODBC Stage or UniVerse Stage dialog box.

Select the data source name and enter your user name and password.

On UniVerse: User name and password are not required for a UniVerse stage using *localuv* as the source.

7. Click the **Inputs** tab.

On the **General** page, select the PRICES table from the **Table name** drop-down list box.

On UniVerse: The table has your account name as a prefix, for example, EXAMPLE.PRICES.

Choose **Clear the table, then insert rows** from the choices in the **Update action** drop-down list box. This lets you reuse the PRICES table, which was populated if you completed Exercise 4.

8. On the **Columns** page, click **Load...** to load the ODBC definition for the PRICES table (or the UniVerse definition if using UniVerse data). The database provided this information during the meta data import that you did in Exercise 12.
9. To see the SQL statements that DataStage uses for this job, click the **View SQL** tab.

Click **OK**. The ODBC or UniVerse stage is now complete.

10. Edit the Transformer stage using the Transformer Editor.

You have only to define the column derivations on the output link. As in Exercise 4, there is no material data transformation in this exercise, and you should make each derivation a straight mapping of source to target.

Notice though that the order of the input and output columns is different; ensure that you map the correct columns. The derivation for column PRODID is DSLink3.PRODID, for column STARTDATE it is DSLink3.STARTDATE, and so on.

When you have set the column derivations, click **OK** to accept the transformer settings and to exit the Transformer Editor.

The Transformer stage is now complete.

11. Save the job, then compile it.
12. Open the DataStage Director and validate the new job.
13. Run the job. The PRICES table is loaded with data from the multivalued file.

To examine the table, return to the job Exercise13 in the DataStage Designer, open the ODBC (or UniVerse) Stage dialog box, then click **View Data...** on the **Inputs** page. The Data Browser displays the data in the same layout as shown in step 5, except that the order of the columns is different.

You have successfully created a job to take records with multivalued data from a UniData or UniVerse file, and loaded that data in 1NF form into your data mart.

Summary

In this chapter you created a sample multivalued hashed file that contained data in nonfirst-normal form (NF²), then you created meta data from this file. You used the meta data in a job that extracted the data from the hashed file and loaded it into a relational database table in first normal form (1NF). The job included a UniData stage or Hashed File stage, which you used for the first time in this tutorial.

9

National Language Support

The exercises in this chapter demonstrate some of the National Language Support (NLS) features of DataStage. You do the following:

- Convert data from the EBCDIC character set to the ASCII character set.
- Use per-column mapping to handle a table that has columns using different character sets.
- Sort data according to the conventions of two different locales and examine the difference between the two resulting outputs.

To do the exercises, you need DataStage installed with NLS.

National Language Support in DataStage

DataStage can handle data in many different character sets through built-in National Language Support (NLS). DataStage achieves this data handling capability by using an internal character set based on UNICODE for all data handling and internal functions. UNICODE is a 16-bit character set capable of encoding characters from most major languages and it is used as a worldwide character-encoding standard. DataStage maps incoming data from whatever character set it is in to this internal representation. Similarly, outgoing data can be mapped from the internal representation to whatever character set is required.

A default external character set is defined for each project. In DataStage, character sets are called maps. Within each job, the default map may be overridden for the job as a whole, for individual stage instances, and for individual columns within tables.

Jobs are neither restricted to the same character set for all of the source and target stages, nor to the character set of the underlying operating system. Multiple character sets may be in use at one time.

This means, for example, that a job could read data from a Korean (KSC) ODBC source and write Japanese (JIS) sequential files. It also means that DataStage can be used for moving data between mainframe database character sets, without changing the meaning of the data at all. Imported meta data, such as table and column definitions, can be in any supported character set. It is easy with the right tools.

In DataStage, national conventions for numbers, dates, times, and currency are known as locales. A default locale is set for each project during installation. By setting locales for a project or job, you can ensure that DataStage uses the appropriate conventions for the territory the application is used in. The conventions define:

- The language and character set for the territory
- How numbers, dates, times, and currency data are formatted
- Which characters are alphabetic, and how they should be sorted

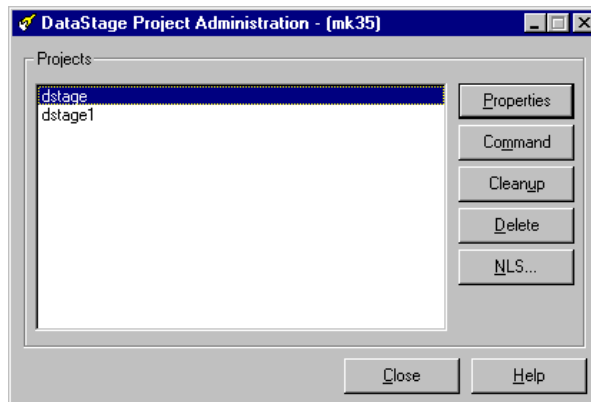
Exercise 14: Convert from One Character Set to Another

This exercise shows NLS support at its simplest: how to convert data from one character set to another. In this case you will be converting from an EBCDIC character set to an ASCII character set. You will be adding some extra items to the PRODUCTS table, using data from an EBCDIC text file.

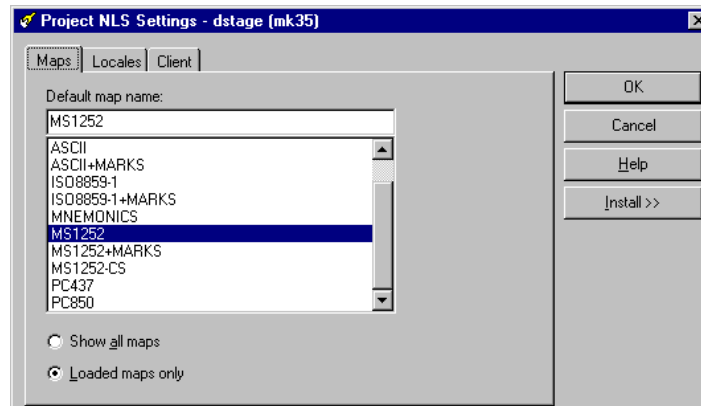
As the exercise uses a character map that is not normally loaded with DataStage, the first step is to install the map using the DataStage Administrator. You have not used the DataStage Administrator in previous exercises, but everything you need to know to use it in this exercise is described here.

To install the map for the EBCDIC character set:

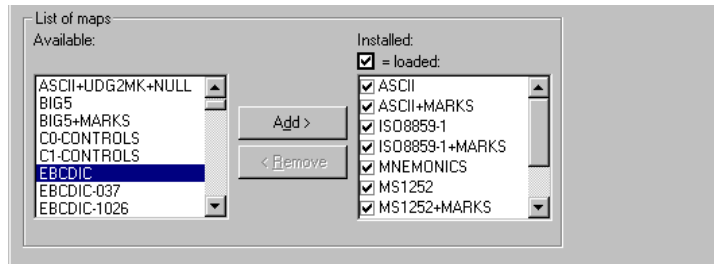
1. Choose **Start ► Programs ► Ardent DataStage ► DataStage Administrator** to start the DataStage Administrator. After you have filled in the Attach to DataStage dialog box, the DataStage Project Administration dialog box appears:



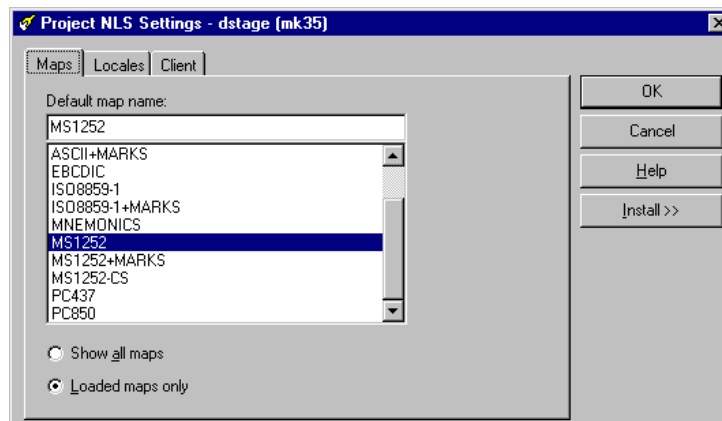
2. Click the **NLS...** button. The Project NLS Settings dialog box appears with the **Maps** page displayed. This is where you load the EBCDIC map.



- Click the **Install>>** button. The dialog box expands to show a list of maps.



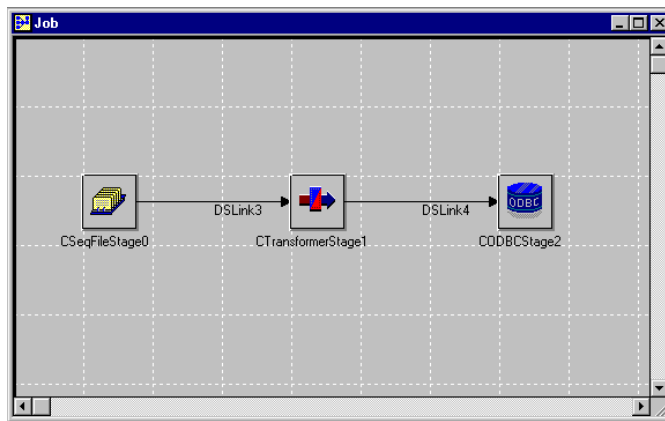
- Select EBCDIC from the list of **Available** maps on the left, then click the **Add>** button. The EBCDIC map is added to the **Installed** list on the right, but before you can use it you need to stop and restart the DataStage server engine. This is because character set maps are only actually loaded by the server engine when it starts up. The easiest way to stop and restart the server is to close any DataStage clients and reboot the computer.
- When you have restarted, open the DataStage Administrator again, and click **NLS...** to go to the **Maps** page of the Project NLS Settings dialog box. Check that EBCDIC now appears in the list of maps.



- Click **OK**, and once you have returned to the DataStage Project Administration dialog box, click **Close** to exit the DataStage Administrator.

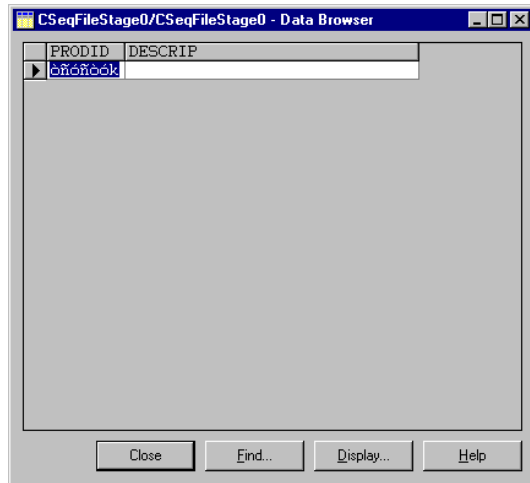
The EBCDIC map is now available for use from DataStage. Next, you go to the DataStage Designer and set up a new job:

1. Open the DataStage Designer and create a new job. Save the job as Exercise14.
2. From left to right, add a Sequential File stage, a Transformer stage, and an ODBC or UniVerse stage.

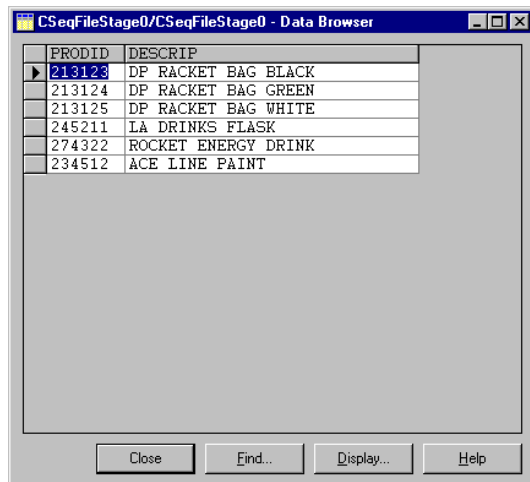


3. Edit the Sequential File stage:
 - a. Enter `c:\Ardent\DataStage\Tutorial` in the **Directory where files are held** field on the **General** page.
 - b. Click the **NLS** tab to view the **NLS** page. You will see that the stage is set to use the default project character set map, e.g., MS1252, which handles ASCII characters. Don't change anything yet.
 - c. Click the **Outputs** tab. Enter `ebcdprod.txt` in the **File name** field. This file contains updates to the PRODUCTS table, but in the EBCDIC character set.
 - d. Click the **Columns** tab. There is no sequential meta data for the PRODUCTS table, but you can load the meta data from the existing PRODUCTS table definition under the ODBC or UniVerse branch of the Repository.

- e. Once you have loaded the table definition, click the **View Data...** button in the **Columns** page. The data shown is nonsense, because DataStage is viewing EBCDIC data using an ASCII character set map. Close the Data Browser and return to the **NLS** page of the Sequential File **Stage** page.



- f. Select the EBCDIC map from the **Map name to use with stage** list. This tells DataStage that this stage uses the EBCDIC map rather than the default project map.
- g. Go back to the **Outputs** page and click the **View Data...** button. This time you should see six new products that will be added to the PRODUCTS table.



4. Edit the ODBC stage:
 - a. From the **General** page, select the DSN you are using for the exercises as the **Data source name** from the drop-down list. Fill in the **User name** and **Password** fields. This time you will be using the project default character set map, so there is no need to visit the **NLS** page.
 - b. Click the **Inputs** tab and on the **General** page, select **PRODUCTS** from the **Table name** drop-down list and **Insert new or update existing** from the **Update action** drop-down list. This ensures you will add to the existing **PRODUCTS** table rather than overwrite it.
 - c. On the **Columns** page, load the table definitions for the **PRODUCTS** table from the **ODBC** (or **UniVerse**) branch.
5. Now you need to edit the Transformer stage. In this exercise the Transformer stage acts as a simple link between the Sequential File stage and the ODBC or UniVerse stage. Open the Transformer Editor and link the columns together from left pane to right pane.

If you save, compile and run the job, you will add the extra six products to the **PRODUCTS** table.

Exercise 15: Use Per-Column Mapping

This exercise demonstrates the use of per-column mapping. Per-column mapping allows different columns in the same table to use different character sets. Main-frame databases in Japan and other countries sometimes contain records where the data consists of single- and double-byte characters, and a mixture of single- and double-byte characters with control codes to switch between them. Data of all three types can exist in a single table. DataStage allows each column in a table to be encoded using a different character set.

Per-column mapping is normally used on an input stage to convert all the source data into the same character set, but it may also be used on an output stage.

In this exercise you read in a sequential file with four columns, each in a different character set. You then produce a sequential file with all four columns in the same character set. The exercise uses the following character sets:

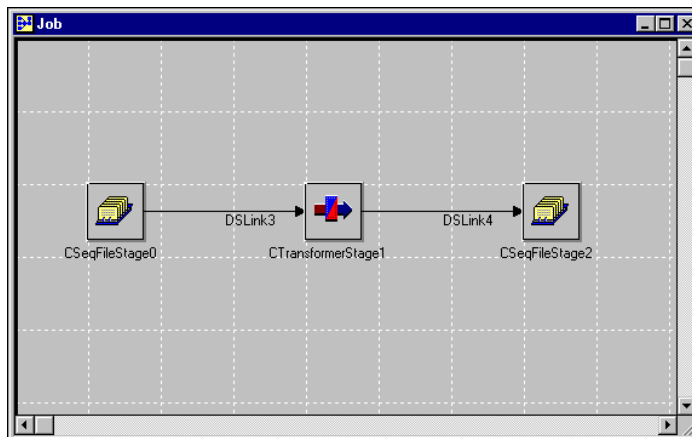
- MS1252 – Latin character set, ASCII.
- EBCDIC – IBM character set.
- MS932 – Japanese character set, based on ASCII.

- JPN-EBCDIC-IBM83 – Japanese character set which uses shift characters to change between single-byte EBCDIC characters and double-byte Japanese characters.

The first thing you need to do is to open the DataStage Administrator and ensure all these character sets are loaded and available in DataStage. Exercise 14 guides you through the process of using the Administrator to add new character set maps. Don't forget to stop and restart your DataStage server engine to enable it to load the new maps.

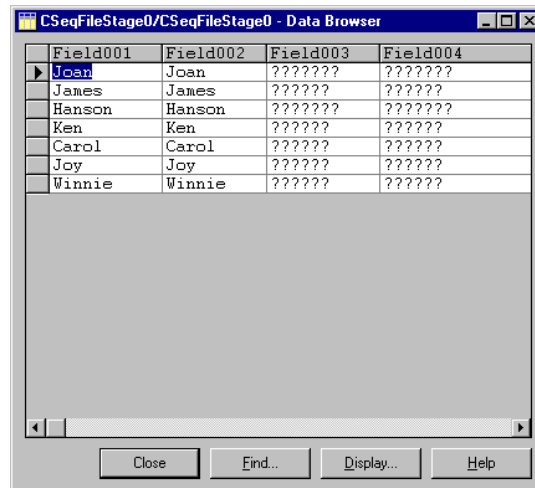
Once all the required maps are available for use within DataStage, go to the DataStage Designer and set up a new job:

1. Create a new job. Save the job as Exercise15.
2. From left to right, add a Sequential File stage, a Transformer stage, and another Sequential File stage.



3. Edit the source Sequential File stage:
 - a. Enter `c:\Ardent\DataStage\Tutorial` in the **Directory where files are held** field on the **General** page.
 - b. Click the **NLS** tab to view the NLS page and select the **Allow per-column mapping** check box. The project default character set map, e.g., MS1252, is currently selected for the stage and is used for any delimiters in the Sequential File data source.
 - c. Click the **Outputs** tab. Enter `percol.txt` in the **File name** field on the **General** page.

- d. Go to the **Columns** page. You are going to enter the table definitions manually on this page. You'll notice that an extra attribute, **NLS Map**, has appeared for each column (you need to scroll across to the right to see this). You will set **NLS Map** to the character set map for each column. If no character set map is specified, the project default map is used. Define four columns as follows:
 - Field001, Key = No, SQL type = Varchar, Length = 255, display = 20, NLS Map = ASCII
 - Field002, Key = No, SQL type = Varchar, Length = 255, display = 20, NLS Map = EBCDIC
 - Field003, Key = No, SQL type = Varchar, Length = 255, display = 20, NLS Map = MS932
 - Field004, Key = No, SQL type = Varchar, Length = 255, display = 20, NLS Map = JPN-EBCDIC-IBM83
- e. Click **View Data...** to look at the input sequential file. You will be able to read the first two columns, but on non-Japanese machines the third and fourth columns contain question marks. This is because your machine cannot display Japanese characters. Don't worry, the DataStage job can still read the file.

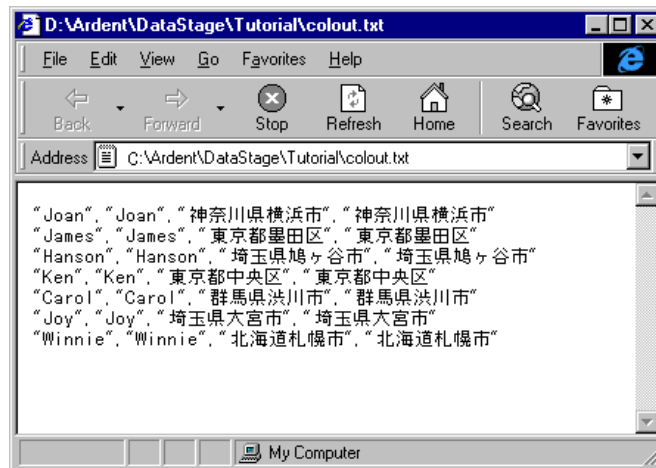


4. Edit the destination Sequential File stage:
 - a. Enter **c:\Ardent\DataStage\Tutorial** in the **Directory where files are held** field on the **General** page.

- b. Click the **NLS** tab to view the **NLS** page. Select MS932 as the character set map for this stage. Do not select per-column mapping this time; all columns will be converted into the Japanese MS932 map before being written to the output file.
 - c. Go to the **Outputs** page. Enter `colout.txt` in the **File name** field on the **General** page. On the **Columns** page enter the table definition manually as follows:
 - Field001, Key = No, SQL type = Varchar, Length = 255, display = 20
 - Field002, Key = No, SQL type = Varchar, Length = 255, display = 20
 - Field003, Key = No, SQL type = Varchar, Length = 255, display = 20
 - Field004, Key = No, SQL type = Varchar, Length = 255, display = 20There is no **NLS Map** field in this column definition because you did not select **Allow per-column mapping** on the **NLS** page.
5. Now you need to edit the Transformer stage. In this exercise the Transformer stage acts as a simple link between the input Sequential File stage and the output Sequential File stage. Open the Transformer Editor and link the columns together from left pane to right pane.

If you save, compile and run the job, you produce a single file called *colout.txt* which contains all the data in a single character set, MS932.

If you are not working on a Japanese machine, you can try using Internet Explorer to view *colout.txt*. Open the file in the browser, right-click in the text window and choose **Language ► Japanese (Auto Select)** from the shortcut menu. If the Japanese option is not available in this menu, it is easy to download Japanese Language Support from the Microsoft product updates Web site.



Exercise 16: Sorting Under Different Locales

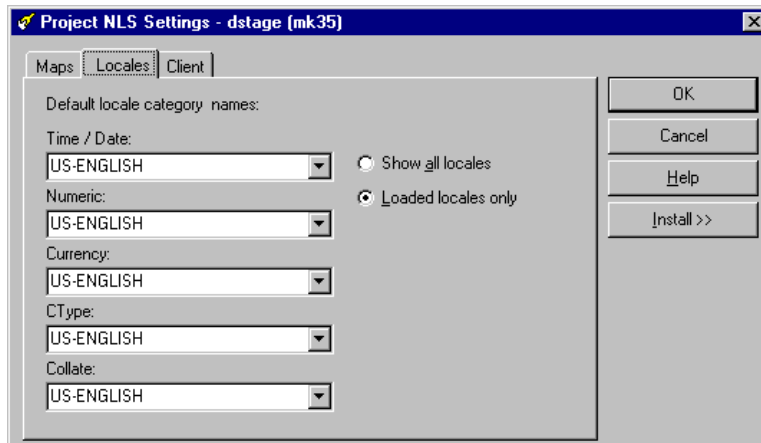
This exercise demonstrates the use of locales within DataStage. Locales provide the means by which language-specific and country-specific conventions can be set. This can affect things as diverse as the format of dates and the order that data is sorted in. You can apply different locales to different conventions on a per-job basis.

In this exercise you explore the difference between the sorting conventions used in the US-English locale, and those used in the French locale. You create a job that compares the values in two columns and states what order they sort in. You run the job once using US English conventions, and once using French conventions.

The exercise requires that the MS1252 character set map and the US-English and FR-French locales are loaded. You can expect the MS1252 character set to be loaded by default on most systems, but you will have to use the DataStage Administrator to ensure that the correct locales are loaded.

1. Open the DataStage Administrator. The DataStage Project Administration dialog box appears.
2. Click the **NLS...** button. The Project NLS Settings dialog box appears with the **Maps** page displayed. Click the **Locales** tab to go to the **Locales** page. You can

use any of the drop-down lists to see which locales are currently loaded into DataStage.

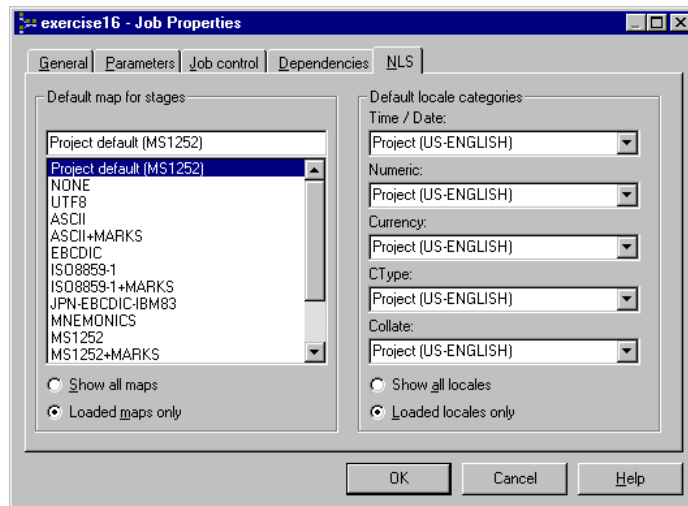


3. If either US-English or FR-French are not loaded, click the **Install>>** button. The dialog box expands to show a list of locales. Loading extra locales is very similar to loading extra maps as described on page 9-3. You select the missing locales from the **Available** list on the left and click **Add>**, then you need to stop and start the DataStage server engine so that it can load the new locales.

Once you have made sure that the required map and locales are loaded, go to the DataStage Designer and set up a new job.

1. Create a new job. Save the job as Exercise16.

2. Open the DataStage Designer and choose **Edit ► Job Properties**. The Job Properties dialog box appears. Go to the NLS page and ensure that the Collate locale is set to US-English.



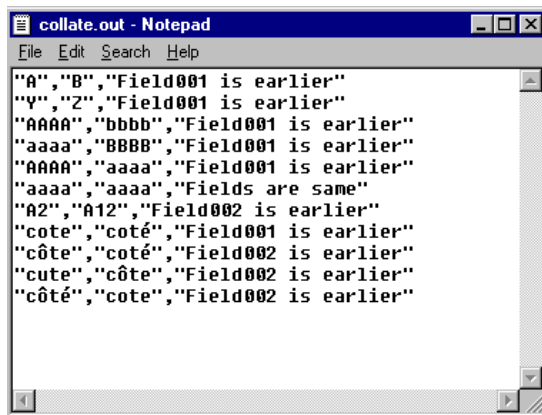
3. From left to right, add a Sequential File stage, a Transformer stage, and another Sequential File stage.
4. Edit the source Sequential File stage:
 - a. Enter **c:\Ardent\DataStage\Tutorial** in the **Directory where files are held** field on the **General** page.
 - b. Click the **Outputs** tab. Enter **collate.txt** in the **File name** field on the **General** page.
 - c. Go to the **Columns** page. Load the table definitions from COLLATE in the Sequential branch of the Repository table definitions.
5. Edit the destination Sequential File stage:
 - a. Enter **c:\Ardent\DataStage\Tutorial** in the **Directory where files are held** field on the **General** page.
 - b. Click the **Outputs** tab. Enter **collate.out** in the **File name** field on the **General** page.

- c. Go to the **Columns** page. Load the table definitions from COLLATE and add an extra column to the table definitions as follows:
 - Result, Key = No, SQL Type = Varchar, Length = 10, nullable = No, Display = 10
6. Next you need to edit the Transformer stage. Link the Field001 and Field002 columns together from left pane to right pane. Define the derivation of the Results field by double-clicking its derivation field. Enter the following derivation into the Expression Editor:

```
if DSLinkx.Field001 < DSLinkx.Field002
then 'Field001 is earlier'
else if DSLinkx.Field001 > DSLinkx.Field002
then 'Field002 is earlier'
else 'Fields are same'
```

DSLinkx is the name of the link from your source sequential file.

If you save, compile and run the job, you produce a single file called *collate.out*. Use an editor such as Notepad to look at the file. You should see something like:

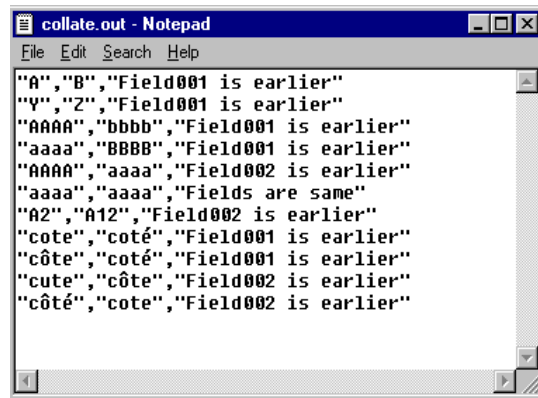


```
collate.out - Notepad
File Edit Search Help
"A","B","Field001 is earlier"
"Y","Z","Field001 is earlier"
"AAAA","bbbb","Field001 is earlier"
"aaaa","BBBB","Field001 is earlier"
"AAAA","aaaa","Field001 is earlier"
"aaaa","aaaa","Fields are same"
"A2","A12","Field002 is earlier"
"cote","coté","Field001 is earlier"
"côte","coté","Field002 is earlier"
"cute","côte","Field002 is earlier"
"côté","cote","Field002 is earlier"
```

Save the file under a different name. Next you are going to recreate this file using the French collate conventions.

1. In the DataStage Designer, choose **Edit ► Job Properties** again. The Job Properties dialog box appears. Go to the NLS page and set the Collate locale to FR-French.
2. Save, compile, and run the job.

Open the new *collate.out* file. This file should look something like:



```
"A","B","Field001 is earlier"
"Y","Z","Field001 is earlier"
"AAAA","bbbb","Field001 is earlier"
"aaaa","BBBB","Field001 is earlier"
"AAAA","aaaa","Field002 is earlier"
"aaaa","aaaa","Fields are same"
"A2","A12","Field002 is earlier"
"cote","coté","Field001 is earlier"
"côte","coté","Field001 is earlier"
"cute","côte","Field002 is earlier"
"côté","cote","Field002 is earlier"
```

Compare the two files to see the differences between them. These demonstrate the differences between the US-English and French collation. Firstly, the accent ordering is different, as shown by the *côte* and *coté* tests. Secondly, when two words differ only in case, the French convention puts the lowercase word first, whereas the US-English convention puts the uppercase word first.

Summary

In this chapter you converted data from the EBCDIC character set to the ASCII character set. You explored the use of per-column mapping by taking data from a table whose columns each used a different character set and outputting it to a table where all columns used the same character set. Finally, you compared the effects of setting different locales on the sort order of data.

10

Additional Features

This chapter briefly describes some of the additional features of the DataStage tool set because we want you to know more about what the DataStage tools can do.

Plug-In Stages

Plug-ins are written to perform specific tasks that the built-in stages do not support. A plug-in consists of a set of routines that access external databases and/or perform complex programming. Passive plug-in stages can be written that support meta data import, enabling the DataStage Manager to browse and retrieve meta data definitions from external sources.

You must have a thorough knowledge of C to design and develop a plug-in.

Two plug-ins are supplied with DataStage:

- BCPLoad
- Orabulk

BCPLoad Stages

The BCPLoad stage is a passive plug-in stage supplied by Ardent. It bulk loads data into a single table in a Microsoft SQL Server (Release 6.5 or 7.0) or Sybase (System 11.5) 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.

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.

Transforms

DataStage has a comprehensive set of built-in transforms that handle many of your basic data-manipulation requirements. It also provides an easy-to-use tool that lets you create your own transforms.

Built-In Transforms

You can view the built-in transforms using the DataStage Manager. They are grouped in two categories in the project tree, according to the type of data they return:

- **Dates** – The Dates transforms mostly convert input data in *xx.TAG* format to a numeric internal data, or convert numeric internal dates to strings in *xx.TAG* format. This category also includes transforms that convert data between *TIMESTAMP* format and the internal date format.
- **String** – The primary function of these transforms is to manipulate input strings in a variety of ways, but they also include transforms that return a string from a number.

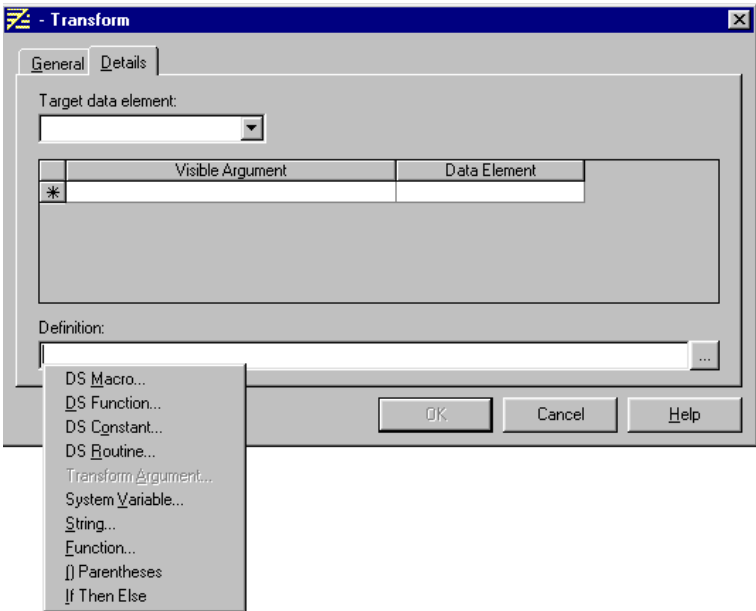
Double-click a transform in the DataStage Manager project tree to view a description of the function of a transform, and its source and target data elements.

Refer to *DataStage Developer's Guide* for a description of each built-in transform.

Custom Transforms

If the built-in transforms are not suitable or you want a transform to act on a specific data element, you can create custom transforms. Custom transforms are created in the DataStage Manager.

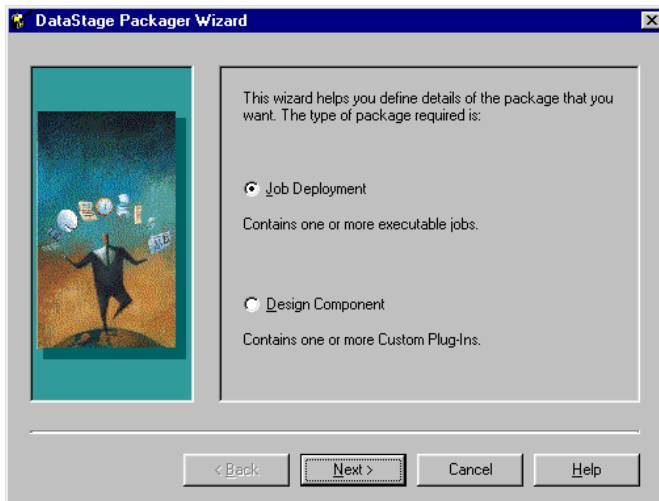
You can easily organize your transforms in branches under the main **Transforms** branch in the DataStage Manager project tree. The Transform dialog box lets you specify the branch and the transform name, select the target data element, and specify the source arguments and data types. You then define the behavior of the transform using the DataStage Expression Editor, which is invoked automatically from the Transform dialog box. The following example shows the Transform dialog box and the menu presented by the Expression Editor.



When you have created the transform, it becomes available for use from within the Transformer Editor. Refer to *DataStage Developer's Guide* for details of how to create a custom transform.

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 DataStage Manager.



When you run the Packager Wizard, you must select the type of package to create:

- **Job Deployment.** This package 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.
- **Design Component.** This package contains plug-in definitions and associated DLLs.

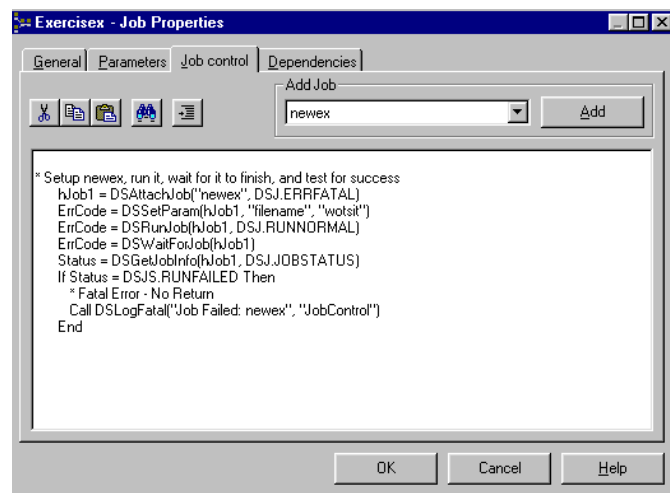
The Packager Wizard then prompts for the package name and the directory in which it should be created. It then automatically displays a list from which you select the jobs or plug-ins that you want to include in the package. For a Job Deployment package, this list box displays all the released jobs. For a Design Component package, it displays all the plug-ins under the **Stage Types** branch in the Repository (except the built-in ones supplied with DataStage).

Job Control Routines

A job control routine enables you to control other jobs from within a single job. There are two possible approaches:

- Set up a routine within the job that validates, runs, stops, and schedules a set of other jobs.
- Set up a job whose only function is to control other jobs.

Job control routines are defined from the DataStage Designer. You open the job in which you want to create the routine, then display the job properties. The Job Properties dialog box has a **Job control** page that enables you to cut, copy, paste, search, and format the code for the job control routine. The main part of the page consists of a multiline text box with scroll bars. This box displays the code for the routine.



You can add a job to the job control routine by selecting it from a list of all the jobs in the current project. When you select a compiled job, the Job Run Options dialog box lets you specify any parameters or run-time limits that you want to apply when the job runs. DataStage automatically adds job control code for the job to the text box, as shown above. This code sets any job parameters and limits, runs the job, waits for it to finish, then tests for success.

Note: In this example the job called in the job control routine runs before anything else in the current job. If the called job aborts, the current job also aborts, without running any further steps.

Each job control routine is built from a set of BASIC functions that DataStage provides specifically for this purpose. For more information about these functions, see the DataStage Developer's Help.

The **Job control** page provides a basic editor that you can use to construct a routine by typing the jobs, parameters, run-time limits, and so on directly into the code in the text box.

For an example of a job control routine, refer to *DataStage Developer's Guide*.

Reports

DataStage provides a flexible reporting tool that allows you to generate reports at various levels within a job design, for example, an entire job, a single stage, or a set of stages. Developers can thus build hard-copy records of their job designs, share these designs with other developers, or use them to work offline from the DataStage Designer.

The reports reflect the structure of components in a job and thus help the developer associate the information in the report with that presented by the DataStage user interface. A job listing, for example, details the job stages, then the links and, for each link, the definitions of the columns on that link.

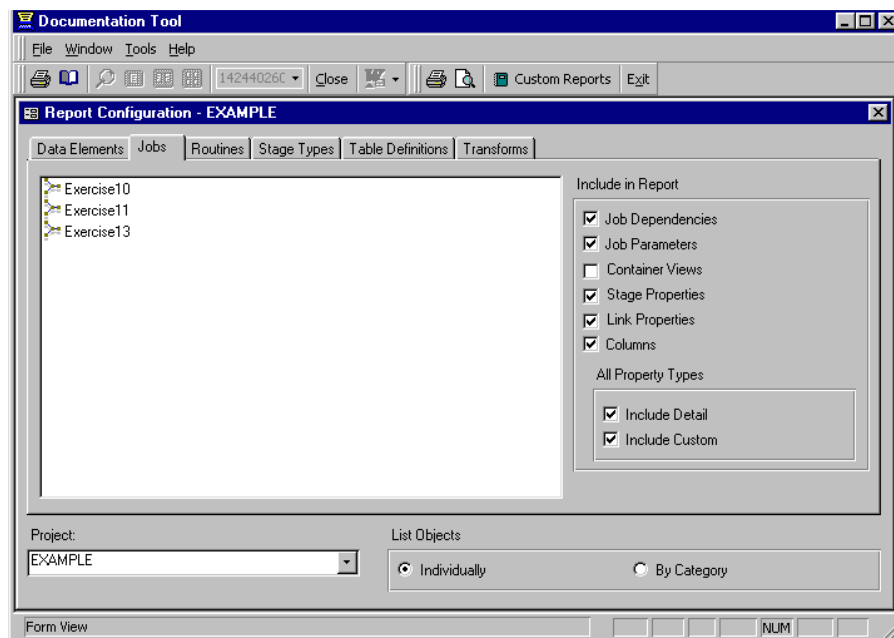
This section summarizes the features provided by the Reporting Tool. For a detailed description of the Reporting Tool and the procedures for using it, refer to *DataStage Developer's Guide*.

The report information generated by DataStage is stored in a relational database on the DataStage client. You can use this information to print a report, or write a report to a file, or you can use a third-party tool to interrogate the information.

The DataStage client contains a Microsoft Access database in which the report information can be stored. This database offers predefined report formats, and you can extend or customize it. However, you can also target one of a number of other popular SQL databases. Support scripts to create the reporting schema in these databases are installed on the client machine. Refer to *DataStage Developer's Guide* for further information.

You use the Reporting Tool dialog box to update the reporting database. It lets you update the entire project or select the objects that you want to update. There are also options to make adjustments to factors that affect the updates. For example, you can insert carriage returns into potentially long properties such as the Description property.

The Documentation Tool, which is invoked from the Reporting Tool dialog box, lets you obtain a predefined report from the Microsoft Access database. In the Documentation Tool dialog box you choose the project on which you want to report and then define your report requirements. The Report Configuration form lists the different types of object in the reporting database, and lets you list these by category or individually. Developers can filter the report to focus on aspects of the design in which they are interested, or to limit the level of detail.



When you have defined your report requirements, you can preview the finished report, and print it if the preview is satisfactory.

This dialog box also provides a Custom Reports function, which enables you to choose from reports that are additional to those supplied with DataStage.

11

Summary

This chapter summarizes the main features of DataStage and recaps what you learned during the course of this tutorial.

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.

Recap of the Exercises

Right from the start, you learned how to use the DataStage tool set.

Although DataStage can support much more advanced scenarios than appeared in this tutorial, a lot of material was covered in a short time.

The following list shows what was covered in the exercises:

1. Creating and editing jobs.
 2. Saving and compiling jobs.
 3. Validating and running jobs.
 4. Creating and editing Sequential File stages.
 5. Creating and editing Transformer stages.
 6. Creating and editing ODBC or UniVerse stages.
 7. Using the Data Browser to view the data in files and tables.
 8. Creating a time dimension.
 9. Working with multiple data sources and targets:
 - Combined in a single stage
 - Separated into multiple stages
 10. Using row constraints to redirect data output.
 11. Loading a fact table.
 12. Creating and editing Aggregator stages.
 13. Creating and loading summary tables.
 14. Creating and editing meta data from:
 - Comma-delimited sequential files
 - ODBC tables
 15. Importing meta data into the Repository.
 16. Running the job design debugger.
- Specifically for UniData or UniVerse users:
17. Creating data from UniData or UniVerse hashed files.
 18. Creating and editing UniData or Hashed File stages.
- Specifically for DataStage NLS users:
19. Converting data from one character set to another.
 20. Dealing with tables where different columns are in different character sets.
 21. Sorting data according to the conventions of two different locales.

During the tutorial you also learned how to navigate the DataStage user interface in:

- The DataStage Manager and Repository
- The DataStage Designer
- The DataStage Director
- The DataStage Administrator

And you found out something about:

- Plug-in stages
- Built-in and custom transforms
- The Packager Wizard
- Job control routines
- The Reporting Tool
- The Documentation Tool
- National Language Support

You worked on some fairly complex examples, but somehow it did not seem that difficult. It is easy with the right tools.

A

Sample Data Definitions

This appendix contains table and column definitions for the data used in the exercises.

The following tables contain the complete table and column definitions for the sample data. They illustrate how the properties for each table should appear when viewed in the Repository.

For UniVerse, ORACLE and Sybase users, the data types may differ slightly from those specified for ODBC tables. If you use UniVerse, for example, you see Date rather than Timestamp as the SQL types in the table definitions. Do not change the table definitions generated during the tutorial setup process (see “Installing Sample Data” on page 2-10) unless you are instructed to do so as part of a tutorial exercise. For example, if you are working in an Informix or ORACLE environment, you are instructed to change the following elements of column definitions that have an SQL type of Timestamp:

	Length	Scale	Display
On Informix:	13	0	13
On ORACLE:	19	0	19

This adjustment to the meta data is specific to the tutorial.

ITEMS and PRICES each have two definitions. One applies to the sequential file, while the other applies to the ODBC table. They are not the same, which allows you to do exercises that extract data from the sequential files, transform that data, then load it into the corresponding tables.

MVPRICES is a multivalued hashed file used with UniData and Hashed File stages. It is used to demonstrate the way in which DataStage handles multivalued data.

The sequential file definitions are listed first, in alphabetical order, followed by the hashed file definition:

Sequential File: COLLATE.TXT (NLS exercises only)

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
Field002	No	VarChar	255		No	8	
Field002	No	VarChar	255		No	8	

Sequential File: EBCDPROD.TXT (NLS exercises only)

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
PRODID	Yes	Decimal	6		No	8	Number
DESCRIP	No	Char	30		Yes	30	DATE.TAG

Sequential File: ITEMS.TXT

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
QTY	No	Integer	4		Yes	4	Number
PRODID	No	Integer	6		Yes	6	Number
ORDID	Yes	Integer	4		No	4	Number
ITEMTOT	No	Integer	10		Yes	10	Number
ITEMID	Yes	Integer	4		No	4	Number
ACTUALPRICE	No	Integer	10		Yes	10	Number

Sequential File: PERCOL.TXT (NLS exercises only)

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	NLS Map
Field002	No	VarChar	255		No	20	ASCII
Field002	No	VarChar	255		No	20	EBCDIC
Field002	No	VarChar	255		No	20	MS932
Field002	No	VarChar	255		No	20	JPN-EBCDIC-IBM83

Sequential File: PRICES.TXT and MYPRICES.TXT

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
STDPRICE	No	Decimal	4	2	Yes	4	Number
STARTDATE	Yes	Char	10		No	10	DATE.TAG
PRODID	Yes	Decimal	6		No	6	Number
MINPRICE	No	Decimal	4	2	Yes	4	Number
ENDDATE	No	Char	10		Yes	10	DATE.TAG

Hashed File: MVPRICES

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
PRODID	Yes	VarChar	10		No		
STDPRICE	No	Integer	10		Yes	11	
MINPRICE	No	Integer	10		Yes	11	
STARTDATE	No	VarChar	10		Yes		
ENDDATE	No	VarChar	10		Yes		

Next the ODBC table definitions are listed in alphabetical order:

ODBC Table: CUSTOMERS

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
CUSTID	Yes	Decimal	6		No	8	Number
NAME	No	Char	45		Yes	45	String
ADDRESS	No	Char	40		Yes	40	String
CITY	No	Char	30		Yes	30	String
STATE	No	Char	2		Yes	2	String
ZIP	No	Char	9		Yes	9	String
AREA	No	Decimal	3		Yes	5	Number
PHONE	No	Char	9		Yes	9	String
REPID	No	Decimal	4		No	6	Number
CREDITLIMIT	No	Decimal	9	2	Yes	11	Number
COMMENTS	No	Char	255		Yes	255	String

ODBC Table: EMPLOYEES

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
EMPNO	Yes	Decimal	4		No	6	Number
ENAME	No	Char	10		Yes	10	String
JOB	No	Char	9		Yes	9	String
MGR	No	Decimal	4		Yes	6	Number
HIREDATE	No	Timestamp	23	3	Yes	23	Timestamp
SAL	No	Decimal	7	2	Yes	9	Number
COMM	No	Decimal	7	2	Yes	9	Number

ODBC Table: FACTS

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
ORDERDATE	Yes	Timestamp	23	3	No	23	Timestamp
EMPNO	Yes	Decimal	4		No	6	Number
CUSTID	Yes	Decimal	6		No	8	Number
ORDID	Yes	Decimal	4		No	6	Number
PRODID	Yes	Decimal	6		No	8	Number
QTY	No	Decimal	8		Yes	10	Number
TOTALSALE	No	Decimal	8	2	Yes	10	Number
SHIPDATE	No	Timestamp	23	3	Yes	23	Timestamp

ODBC Table: ITEMS

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
ORDID	Yes	Decimal	4		No	6	Number
ITEMID	Yes	Decimal	4		No	6	Number
PRODID	No	Decimal	6		Yes	8	Number
ACTUALPRICE	No	Decimal	8	2	Yes	10	Number
QTY	No	Decimal	8		Yes	10	Number
ITEMTOT	No	Decimal	8	2	Yes	10	Number

ODBC Table: ORDERS

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
ORDID	Yes	Decimal	4		No	6	Number
ORDERDATE	No	Timestamp	23	3	Yes	23	Timestamp
COMMPLAN	No	Char	1		Yes	1	String
CUSTID	No	Decimal	6		No	8	Number
SHIPDATE	No	Timestamp	23	3	Yes	23	Timestamp
TOTAL	No	Decimal	8	2	No	10	Number

ODBC Table: PRICES (and MYPRICES)

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
PRODID	Yes	Decimal	6		No	8	Number
STARTDATE	Yes	Char	10		No	10	DATE.TAG
STDPRICE	No	Decimal	8	2	Yes	10	Number
MINPRICE	No	Decimal	8	2	Yes	10	Number
ENDDATE	No	Char	10		Yes	10	DATE.TAG

ODBC Table: PRODUCTS

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
PRODID	Yes	Decimal	6		No	8	Number
DESCRIP	No	Char	30		Yes	30	DATE.TAG

ODBC Table: Q_SALES

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
QUARTER	Yes	Char	6		No	6	QUARTER.-TAG
CUSTID	Yes	Decimal	6		No	8	Number
TOTALSALE	No	Decimal	8	2	Yes	10	Number

ODBC Table: REJECTS

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
ORDERDATE	Yes	Timestamp	23	3	No	23	Timestamp
EMPNO	Yes	Decimal	4		No	6	Number
CUSTID	Yes	Decimal	6		No	8	Number
ORDID	Yes	Decimal	4		No	6	Number
PRODID	Yes	Decimal	6		No	8	Number
QTY	No	Decimal	8		Yes	10	Number
TOTALSALE	No	Decimal	8	2	Yes	10	Number
SHIPDATE	No	Timestamp	23	3	Yes	23	Timestamp

ODBC Table: TIME

Column Name	Key	SQL Type	Length	Scale	Nullable	Display	Data Element
DAY	Yes	Timestamp	23		No	23	Timestamp
MONTH	No	Char	2		Yes	2	MONTH.-TAG
QUARTER	No	Char	6		Yes	6	QUARTER.-TAG
YEAR	No	Char	4		Yes	4	YEAR.TAG

Note: For UniVerse users, the TIME table is called TIME_ because TIME is a reserved word in UniVerse.

Index

A

- active stage 1-3
- Adobe Acrobat Reader vii, xi
- aggregating data 5-9
- Aggregator stage 2-19, 5-9
 - definition 1-7
 - properties 5-10
- Attach to Project dialog box 2-17, 3-3, 6-2

B

- BCPLoad stage 10-1
 - definition 1-7
- before- and after-stage subroutines 5-11
- branches 6-4
- breakpoints, setting 7-3
- built-in routines 6-4
- built-in stages 1-3
 - BCPLoad 10-1
 - Orabulk 10-2
- built-in transforms 6-4, 10-2
- bulk loading data 10-1
- buttons
 - tool palette 3-5
 - toolbar 3-4, 3-11, 3-13

C

- changing
 - link names 4-4
 - object names 4-5
- character set locales 1-4
- character set maps 1-4
 - converting from one to another 9-2
 - definition 1-8
- cleanup script files
 - INF_CLN.SQL 2-12
 - MSS_CLN.SQL 2-12
 - ORA_CLN.SQL 2-12
 - SYB_CLN.SQL 2-12
 - UNI_CLN.SQL 2-12
- client components 1-5
- column definitions A-1
 - definition 1-7
 - entering manually 6-8
- Column Display dialog box 8-6
- Compile Job window 3-12, 4-9
- compiling jobs 3-11, 4-9
- connecting to projects 2-17, 3-3
- constraints on rows 5-7
- Container Input stage 1-3
- Container Output stage 1-3
- Container stage 1-3
 - definition 1-7
- containers 1-3
- converting from one character set map to another 9-2
- Create New Job dialog box 4-3
- creating
 - custom transforms 10-3
 - expressions 4-21
 - jobs 4-3

- meta data 6-1, 6-5, 6-7, 8-3
- sample multivalued file 8-2
- sample tables 2-13
- table definitions 6-1, 6-7
- custom SQL code, writing 4-14
- custom transforms, creating 10-3

D

data

- aggregating 5-9
- bulk loading 10-1
- definitions A-1
- direct access 2-9
- extracting, *see* extracting data
- hashed file 2-9
- multivalued 8-1
- ODBC 2-9
- sample 2-1, 2-10, A-1
- sequential file 2-9
- sources 2-9
- targets 2-9
- transforming 4-1, 4-11, 5-1
- types 2-9
- viewing 7-8, 8-5
- writing, *see* loading data
- Data Browser 3-8, 4-16, 7-8, 8-5, 9-6
 - definition 1-7
- data elements, definition 1-7
- data marts 2-5
- data model 2-1
- data properties 1-2
- data transformations 2-18
- data warehouses 2-5
- DataStage
 - features, summary 11-1
 - overview 1-1
 - terms 1-7
- DataStage Administrator 1-5, 1-6
 - definition 1-7
- DataStage Designer 1-5
 - definition 1-7

- starting from Director 4-9
- DataStage Designer window 3-4
- DataStage Director 1-6, 3-13
 - definition 1-7
 - starting 3-12
- DataStage Director window 3-13
- DataStage export file 2-11, 2-16
- DataStage Manager 1-5, 6-2
 - definition 1-7
 - starting 6-2
- DataStage Manager window 2-18, 6-3
 - toolbar 6-3
- DataStage Project Administration
 - dialog box 9-3
- DataStage Repository, *see* Repository
- DataStage Server 1-4
- DataStage tools 1-1
 - client components 1-5
 - Documentation Tool 10-7
 - jobs 1-1
 - projects 1-1
 - Reporting Tool 10-6
 - server components 1-4
 - stages 1-3, 2-10
 - terms 1-7
- Debug Window 7-5
- debugger toolbar 7-3
- debugging a job design 7-1
- defining
 - file formats 6-10
 - table definitions
 - assisted 6-1, 6-5
 - manually 6-1, 6-7
- delimited files, writing data to 4-3
- Derivation dialog box 5-11
- Designer, *see* DataStage Designer
- developer, definition 1-7
- dialog boxes
 - Attach to Project 3-3, 6-2
 - Column Display 8-6
 - Create New Job 4-3
 - Derivation 5-11

- Edit Breakpoints 7-4
- Hashed File Stage 8-4
- Import Meta Data (ODBC) 6-5
- Import Meta Data (UniData Files) 8-3
- Import Meta Data (UniVerse Files) 8-3
- Import Meta Data (UniVerse Tables) 6-5
- Job Run Options 3-14, 4-9
- ODBC Stage 4-13, 4-19, 8-7
- Open Job 3-6
- Sequential File Stage 4-12, 7-2
- Table Definition 6-7
- UniData Stage 8-4
- UniVerse Stage 4-13, 8-7
- dimension tables 2-6
- direct access data 2-9
- Director, *see* DataStage Director
- display area 6-4
 - updating 3-13
- displaying
 - project items 6-4
 - ToolTips 3-5, 3-11, 3-13, 7-3
- documentation conventions ix
- Documentation Tool 10-7

E

- Edit Breakpoints dialog box 7-4
- editing
 - Hashed File stage 8-4
 - ODBC stage 4-13, 4-19, 5-2, 8-7
 - Sequential File stage 3-7, 4-4, 4-12, 4-18, 7-2
 - Transformer stage 3-9, 4-8, 4-14, 4-19, 5-6, 8-8
 - UniData stage 8-4
 - UniVerse stage 4-13, 4-19, 5-2, 8-7
- entering
 - column definitions 6-8
 - logon settings 3-3, 6-2

- environments, *see* SQL environments
- executable jobs 1-2
- exercises
 - aggregating data 5-9
 - converting from one character set map to another 9-2
 - creating meta data
 - multivalued files 8-1
 - relational database tables 6-5
 - sequential file 6-7
 - debugging a job design 7-1
 - extracting data from a multivalued file 8-4
 - loading data into a relational database table 4-17, 6-10
 - loading data into a sequential file 4-2
 - loading the time dimension table 4-17
 - overview 2-18
 - sample job 3-1
 - sorting under different locales 9-11
 - summary 11-2
 - using multiple sources and targets 5-1
 - using per-column mapping 9-7
- export file 2-11, 2-16
- Expression Editor 4-20, 5-8
- expressions, creating 4-21
- extracting data
 - delimited files 4-11
 - hashed file 8-4
 - multiple data sources 5-1
 - multivalued file 8-4
 - relational database tables 5-1
 - sequential file 3-1, 4-2, 4-11
 - UniData file 8-4

F

- fact tables 2-5
- FACTS table 2-19, 5-6, 5-10, A-4
- files
 - export 2-11, 2-16
 - formats, defining 6-10
 - hashed 1-7, 2-9, 8-1, 8-4
 - log 5-9
 - multivalued 8-1, 8-2, 8-4
 - sequential 2-9, 3-1, 4-2, 7-2
 - SQL script 2-12
- first normal form 8-8
 - definition 1-7
- fixed-length sequential files, extracting
 - data from 4-3
- flat files, *see* sequential files
- functions, syntax 4-21

H

- Hashed File stage 8-4
 - definition 1-8
- Hashed File Stage dialog box 8-4
- hashed files 2-9
 - column definition A-2
 - definition 1-7
 - extracting data from 8-4
 - importing meta data from 8-1
- Help system, one-line help 3-4

I

- icons 6-4
- Import Meta Data (ODBC) dialog box 6-5
- Import Meta Data (UniData Files) dialog box 8-3
- Import Meta Data (UniVerse Files) dialog box 8-3
- Import Meta Data (UniVerse Tables) dialog box 6-5

- importing meta data 6-1
 - hashed file 8-1
 - ODBC tables 6-5
 - sample meta data 2-16
 - sequential file 6-1
 - UniData file 8-1
 - UniVerse tables 6-5
- Informix SQL script files 2-12
 - INF_BLD.SQL 2-12
 - INF_CLN.SQL 2-12
 - starting up environment 2-14
- installing tutorial
 - prerequisites viii
 - sample data 2-10
- introduction, *see* overview

J

- job batches, definition 1-8
- job control routines 10-5
- job design debugger 7-1
 - running 7-3
- job parameters, definition 1-8
- Job Run Options dialog box 3-14, 4-9
- Job Status view 3-13
- Job window 3-4, 3-6, 4-4
- jobs 1-1
 - compiling 3-11, 4-9
 - control routines 10-5
 - creating 4-3
 - debugging 7-1
 - definition 1-8
 - example 3-1
 - importing 2-16
 - executable 1-2
 - packaging 10-4
 - sample 3-1, 3-6
 - saving 4-3
 - validating 3-14

L

- links 1-2, 3-7
 - changing names 4-4
 - input 3-10
 - output 3-10
- loading data
 - into ODBC tables 4-17, 5-1, 5-9
 - into sequential files 3-1, 4-2
 - into time dimension tables 4-17
 - into UniVerse tables 4-17, 5-1, 5-9
- locales 1-4
 - definition 1-8
 - sorting under different 9-11
- log files 5-9
- logon settings
 - entering 3-3, 6-2
 - saving 2-17, 3-3

M

- Manager, *see* DataStage Manager
- maps 1-4
 - definition 1-8
 - per-column mapping 9-7
 - project default 1-4
- meta data 6-1
 - creating manually 6-7
 - creating with assistance 6-5
 - definition 1-8
 - importing 2-16, 6-5
 - multivalued files 8-1
 - ODBC tables 2-16, 6-5
 - UniVerse tables 2-16, 6-5
- Microsoft SQL Server SQL script files 2-12
 - MSS_BLD.SQL 2-12
 - MSS_CLN.SQL 2-12
- Microsoft SQL Server, *see* Microsoft SQL Server SQL script files
- moving tool palette 3-5
- MS SQL Server SQL script files

- starting up environment 2-14
- multiple data sources and targets 2-19, 5-1
- multivalued data 8-1, A-1
- multivalued files
 - creating meta data 8-1
 - extracting data 8-4
 - sample 8-2

N

- National Language Support 1-3
- NLS 9-1
 - converting from one character set map to another 9-2
 - definition 1-8
 - overview 1-3
 - per-column mapping 9-7
 - sorting under different locales 9-11
- nonfirst-normal form 8-1
 - definition 1-8
- normalization 8-5
 - definition 1-8

O

- object names, changing 4-5
- ODBC
 - data 2-9
 - drivers viii, 2-9
- ODBC stage
 - definition 1-8
 - editing 4-13, 4-19, 5-2, 8-7
- ODBC Stage dialog box 4-13, 4-19, 8-7
- ODBC tables
 - definitions A-1
 - extracting data 5-1, 5-9
 - loading data 4-11, 4-17, 5-1, 5-9
 - meta data, importing 2-16, 6-5
- Open Job dialog box 3-6
- operator, definition 1-9

- Orabulk stage 10-2
 - definition 1-9
- Oracle SQL script files 2-12
 - ORA_BLD.SQL 2-12
 - ORA_CLN.SQL 2-12
 - starting up environment 2-15
- overview
 - of DataStage 1-1
 - of exercises 2-18, 11-2
 - of NLS 1-3, 9-1
 - of Reporting Tool 10-6
- P**
- Packager Wizard 10-4
- packaging
 - jobs 10-4
 - plug-ins 10-4
- passive stage 1-3
- per-column mapping 1-4
- plug-in stages 1-3, 10-1
 - definition 1-9
- plug-ins
 - BCPLoad 10-1
 - definitions 10-4
 - distributing to other systems 10-4
 - Orabulk 10-2
 - packaging 10-4
- prerequisites viii
- project default map
 - overriding 1-4
- project items, displaying 6-4
- Project NLS Settings dialog box 9-3
- project tree 6-4
- projects 1-1, 6-4
 - connecting to 2-17, 3-3
- properties
 - Aggregator stage 5-10
 - data 1-2

Q

- Q_SALES table 2-19, 5-9, A-5

R

- rejected rows 5-9
- REJECTS table 2-19, 5-6, A-6
- Report Configuration form 10-7
- Reporting Tool, overview 10-6
- Repository 1-4
 - definition 1-9
 - editing items in 6-10
- routines 6-4
- rows
 - constraints on 5-7
 - rejected 5-9
- running
 - database 2-13
 - job design debugger 7-3
 - SQL scripts 2-13

S

- sample data 2-1
 - creating 2-10
 - definitions A-1
- sample job 3-1
 - configuration 3-7
 - importing 2-16
 - opening 3-6
- sample multivalued file, creating 8-2
- sample tables, creating 2-13
- saving
 - jobs 4-3
 - logon settings 2-17, 3-3
- script files, *see* SQL script files
- Sequential File stage 2-19
 - definition 1-9
 - editing 3-7, 4-4, 4-12, 4-18, 7-2
- Sequential File Stage dialog box 4-12, 7-2

- sequential files 2-9
 - data 2-9
 - definitions A-2
 - extracting data from 3-1, 4-2
 - loading data into 3-1, 4-2
 - meta data
 - entering manually 6-7
 - importing 6-1
 - opening 2-11
- server components 1-4
- Server, *see* DataStage Server
- setting breakpoints 7-3
- sorting under different locales 9-11
- sources, definition 1-9
- SQL environments 2-14
 - starting up 2-13
 - syntax 2-13
- SQL script files 2-12
 - running 2-13
 - viewing 2-12
- SQL*Plus window 2-14
- sqlldr* command 10-2
- stages 1-3, 2-10
 - see also* built-in stages
 - active 1-3
 - Aggregator 1-7, 2-19, 5-9
 - BCPLoad 1-7, 10-1
 - built-in 1-3
 - Container 1-3, 1-7
 - Container Input 1-3
 - Container Output 1-3
 - definition 1-9
 - Hashed File 1-8, 2-20, 8-4
 - in exercises, overview 2-18
 - ODBC 1-8, 2-19, 4-13, 4-19, 5-2, 8-7
 - Orabulk 1-9, 10-2
 - passive 1-3
 - plug-in 1-3, 1-9
 - Sequential File 1-9, 2-19, 3-7, 4-4, 4-12, 4-18, 7-2
 - Transformer 1-9, 2-19, 3-1, 4-1, 4-2, 4-11, 4-19, 5-1, 7-2

- UniData 1-9, 2-20, 8-4
- UniVerse 1-10, 2-19, 4-13, 4-19, 5-2, 8-7
- star schema 2-5
 - diagram 2-8
 - fact table 2-5
- starting
 - DataStage Director 3-12
 - DataStage Manager 6-2
 - SQL environments 2-13
- status bar 3-4
- Sybase SQL script files 2-12
 - starting up environment 2-15
 - SYB_BLD.SQL 2-12
 - SYB_CLN.SQL 2-12
- syntax
 - functions 4-21
 - SQL environments 2-13

T

- Table Definition dialog box 6-7
- table definitions 6-4, A-1
 - creating 6-1, 6-7
 - definition 1-9
- tables
 - dimension 2-6
 - exercise samples A-1
 - fact 2-5
 - FACTS 2-19, 5-6, 5-10, A-4
 - meta data, importing 2-16, 6-5
 - names 6-6
 - ODBC, *see* ODBC tables
 - Q_SALES 2-19, 5-9, A-5
 - REJECTS 2-19, 5-6, A-6
 - sample, creating 2-13
 - TIME 2-7, 4-19, 5-10, A-6
 - time dimension 2-7
 - TIME_, *see* TIME
 - UniVerse, *see* UniVerse tables
- terminology 1-7

- text files 2-11
 - see also* sequential files
- time dimension tables 2-7
 - loading 4-17
- TIME table 2-7, 4-19, 5-10, A-6
- TIME_ table, *see* TIME table
- tool palette 3-5
 - buttons 3-5
 - moving 3-5
- toolbars
 - debugger 7-3
 - Designer 3-4
 - Director 3-13
 - Manager 6-3
 - Transformer Editor 3-11
- ToolTips 3-5, 3-11, 3-13, 7-3
- transform functions, definition 1-9
- Transformer Editor 3-10, 4-8, 4-14, 4-19, 5-6, 8-8
 - definition 1-9
 - Links area 3-10
 - Meta Data area 3-10
 - toolbar 3-11
- Transformer stage 2-19
 - definition 1-9
 - editing 3-9, 4-8, 4-14, 4-19, 5-6, 8-8
- Transformer Stage Constraints dialog box 5-7
- transforming data 4-1
 - in hashed files 8-4
 - in relational database tables 5-1, 5-9
 - in sequential files 3-1, 4-2, 4-11
 - in UniData files 8-4
- transforms
 - built-in 10-2
 - custom 10-3
 - dates 10-2
 - string 10-2
- tutorial
 - installing sample data 2-10
 - prerequisites viii

U

- UNICODE 1-3, 9-1
 - definition 1-9
- UniData files
 - extracting data 8-4
 - hashed file definition A-2
 - meta data, importing 8-1
- UniData stage
 - definition 1-9
 - editing 8-4
- UniData Stage dialog box 8-4
- UniVerse
 - reserved word 2-7
 - starting up environment 2-15
- UniVerse files
 - extracting data 8-4
 - meta data, importing 8-1
- UniVerse SQL script files 2-12
 - UNI_BLD.SQL 2-12
 - UNI_CLN.SQL 2-12
- UniVerse stage
 - definition 1-10
 - editing 4-13, 4-19, 5-2, 8-7
- UniVerse Stage dialog box 4-13, 8-7
- UniVerse tables
 - definitions A-1
 - extracting data 5-1, 5-9
 - loading data 4-11, 4-17, 5-1, 5-9
 - meta data, importing 2-16, 6-5
- updating display area 3-13

V

- validating jobs 3-14
- viewing
 - data 7-8, 8-5, 9-6
 - ToolTips 3-5, 3-11, 3-13, 7-3
 - tutorial SQL script files 2-12
- views
 - Job Status 3-13
 - unnormalized 8-5

W

windows

- Compile Job 3-12, 4-9
- Data Browser 4-16, 7-8, 8-5, 9-6
- DataStage Designer 3-4, 9-5
- DataStage Director 3-13
- DataStage Manager 2-18, 6-3
- Debug 7-5
- Job 3-4, 3-6, 4-4
- SQL*Plus 2-14
- Transformer Editor 3-10
- WISQL32 2-15

WISQL32 window 2-15

writing

- custom SQL code 4-14
- data, *see* loading data

