

Informix Formation

User's Guide

Version 1.4
May 1999
Part No. 000-5519

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™; CBT Store™; C-ISAM®; Client SDK™; ContentBase™; Cyber Planet™; DataBlade®; Data Director™; Decision FastStart™; 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® ROLAP Option; Dynamic Server™ with Universal Data Option™; Dynamic Server™ with Web Integration Option™; Dynamic Server™, Workgroup Edition™; Formation™; Formation Architect™; Formation Flow Engine™; Gold Mine Data Access®; 4GL for ToolBus™; i.Reach™; i.Sell™; Illustra®; Informix®; Informix Enterprise Merchant™; Informix®-4GL; Informix®-InquireSM; Informix-JWorks™; InformixLink®, Informix Session Proxy™; InfoShelf™; Interforum™; I-Spy™; Mediazation™; MetaCube®; NewEra™; ON-Bar™; OnLine Dynamic Server™; OnLine for NetWare®; OnLine/Secure Dynamic Server™; OpenCase®, Orca™; PaVer™; Red Brick® and Design; Red Brick Data Mine™; Red Brick Mine Builder™; Red Brick Decisionscape™; Red Brick Formation™; Red Brick Ready™; Red Brick Systems®, Red Brick Vista™; Regency Support®, Rely on Red BrickSM; RISQL®; Solution DesignSM; STARindex™; STARjoin™; SuperView®, TARGETindex™; TARGETjoin™; The Data Warehouse Company®, The one with the smartest data wins.™; The world is being digitized. We're indexing it.SM; 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.

Documentation Team: Randy Otte, Oakland Editing and Production Team

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 of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202.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

Introduction

Purpose	3
Audience	3
Organization	3
Related Documentation	5
Conventions	6
Syntax Notation	6
Syntax Diagrams	7
Keywords and Punctuation	8
Identifiers and Names	9

Chapter 1

Introduction to Informix Formation

Introduction to Informix Formation	1-3
Overview of the Formation Components	1-4
The Formation Architect	1-7
The Formation Architect User Interface	1-8
The Data Flow Diagram.	1-9
The Formation Flow Engine	1-11

Chapter 2

DocSample1: A Quick Start Guide

In This Chapter	2-3
Opening the DocSample1 Job	2-3
Analyzing the DocSample1 Job	2-5
Defining the Data Stores	2-6
Setting Up the Import and Export Operators	2-9
Joining the Data Flows	2-11
Grouping the Data	2-15
Exporting the Data	2-20
Generating and Building the DocSample1 Job	2-21

	Running and Monitoring the DocSample1 Job	2-23
	Looking at the Executable Files	2-23
	Starting the Job	2-23
	Viewing the Job Log.	2-24
Chapter 3	Using Workspaces and Jobs	
	In This Chapter	3-3
	Using Workspaces	3-3
	Creating, Copying, or Importing a Workspace.	3-4
	Opening a Workspace	3-7
	Exporting a Workspace for Backup	3-7
	Renaming and Deleting Workspaces	3-9
	Using Jobs	3-10
Chapter 4	Defining the Data	
	Defining the Data	4-3
	Overview of Creating Data Definitions	4-3
	The Process of Creating Data Definitions	4-4
	Understanding Data Stores	4-5
	Creating or Modifying Data Definitions Using the Data Definition Wizard	4-8
	Data Definition Naming Rules	4-9
	Considerations When Importing Schema	4-10
	Modifying Data Definitions Within a Data Flow Diagram	4-10
	Using Multibyte Data with Informix Formation	4-11
	Using and Setting Locales.	4-11
	Using the Multibyte Text Data Type	4-14
	Copying Data Between Operating Systems	4-17
Chapter 5	Creating Data Flow Diagrams	
	In This Chapter	5-3
	Overview of Data Flow Diagrams	5-3
	Defining the Input and Output Data Stores	5-6
	Selecting the Import and Export Operators	5-6
	Specifying the Data Store	5-7
	Specifying the Data Location.	5-7
	Selecting the Transformation Operators	5-7
	Connecting the Operators	5-8
	Data Propagation with Import or Export Operators	5-9
	Data Propagation with Transformation Operators	5-9

Providing Operator Parameters	5-10
Validating the Job	5-11

Chapter 6 **Selecting and Using Operators**

In This Chapter	6-5
Overview of Operators	6-6
Operator Ports	6-6
Operator Parameters	6-7
Import and Export Operators	6-8
Importing Data	6-8
Exporting Data	6-9
Synchronizing Imports and Exports Using Execution Order	6-10
Quick List of Import and Export Operators	6-10
File Import Operator	6-11
File Export Operator	6-12
Notes on Using File Import and File Export	6-12
Informix DS Import Operator	6-13
Informix DS Export Operator	6-13
Informix DS AD Import Operator	6-14
Informix DS AD Export Operator	6-15
MSSQL Server Import Operator	6-16
MSSQL Server Export Operator	6-16
ODBC Import Operator	6-17
ODBC Export Operator	6-17
Oracle Server Import Operator	6-18
Oracle Server Export Operator	6-18
Red Brick Import Operator	6-19
Red Brick Export Operator	6-19
Simple Transformation Operators	6-20
Deduplicate Operator	6-20
Filter Operator	6-21
Project Operator	6-22
Sort Operator	6-22
Union Operator.	6-23
Join Transformation Operators	6-24
Join Operator	6-25
Advanced Join Operator.	6-25
Cross Product Join Operator	6-27
Advanced Transformation Operators	6-29
Aggregate Operator	6-29

Cursor Operator	6-30
First Normal Operator	6-31
Group By Operator	6-32
Household Operator	6-33
Program Operator	6-34
Split Operator	6-36
Surrogate Key Operator	6-37
Data Parallelism Operators	6-38
Partition Operator	6-38
Gather Operator	6-39
Performance Considerations	6-40

Chapter 7

Working with Snippets

In This Chapter	7-3
Overview of Visual Snippets	7-3
Defining a Visual Snippet	7-5
Variables.	7-6
Ordering Links	7-6
Standard Functions	7-7
Syntax Functions	7-8
Custom Functions	7-14
Operator Variables	7-15
Defining an Operator Variable	7-15
Limitations	7-16
Data Types Used in Visual Snippets	7-16
The Null Value	7-17
Constants and Lists	7-19
Type Conversions	7-20
Code Snippets	7-21

Chapter 8

Creating and Modifying Flow Engine Machines

In This Chapter	8-3
Introduction to Flow Engine Machines	8-4
Creating or Modifying a Flow Engine Machine	8-5
Defining Temporary Space for Job Execution	8-8
Using the Temporary Space Dialog	8-9
Default Temporary Space Locations	8-10
Guidelines for Setting Temporary Space	8-11

Chapter 9	Generating and Running a Job	
	In This Chapter	9-3
	Generating and Building a Job	9-3
	Selecting the Computer and Directory on Which to Build	9-5
	Generating the Source Code and Runtime Parameters Files	9-5
	Building the Job.	9-8
	Running a Job	9-9
	Setting the Flow Engine Execution Environment	9-10
	Running the Master Job Executable	9-12
	Reviewing and Troubleshooting Job Results	9-13
	Collecting and Reviewing Job Execution Information	9-15
	Using the Log Utility	9-16
	Viewing Job Execution Information	9-21
	Understanding Log Messages	9-24
	Using Log Files	9-26
	Modifying the Formation Log Utility Configuration	9-27
	What Happens When the Log Utility is Not Running.	9-29
	Common Job Problems	9-30
	Database Not Available	9-30
	C++ Reserved Words Were Used as Identifiers in the Job	9-31
	Invalid Flow Engine Machine Used	9-31
	Invalid Field Delimiter Used in a Data Store	9-32
Chapter 10	Advanced Job Topics	
	In This Chapter	10-3
	Examining and Modifying Runtime Parameter Files	10-3
	Examining rbf_machine.dat	10-3
	Examining and Modifying rbf_profile.dat.	10-4
	Transferring Data Between Executing Jobs	10-6
	Using External Libraries in a Job	10-7
	Creating an External Function.	10-7
	Call the External Function from a Visual Snippet	10-9
	Specify Build Options for the External Function	10-9

Appendix A	DocSample2: An Extended Example of a Data Flow Diagram
Appendix B	Summary of Supported Data Types
	Glossary
	Index

Introduction

Purpose.	3
Audience	3
Organization	3
Related Documentation	5
Conventions	6
Syntax Notation	6
Syntax Diagrams	7
Keywords and Punctuation	8
Identifiers and Names	9

Purpose

The *Informix Formation User's Guide* provides a description of how to use the overall Informix Formation™ product, including two hands-on example jobs that take the user from the first to the last step in creating a data integration job.

Audience

The *Informix Formation User's Guide* is written for data warehouse professionals, such as data warehouse designers or administrators. These professionals are typically responsible for designing the integration of data into a database or updating a database or warehouse. This book assumes that the readers are familiar with the operating system and databases that they are using in conjunction with Informix Formation.

Organization

This guide is organized as follows:

[Chapter 1, “Introduction to Informix Formation,”](#) introduces the Informix Formation product and some key concepts to its use.

[Chapter 2, “DocSample1: A Quick Start Guide,”](#) shows you how to create and generate a data integration job with Formation using an existing sample job as a point of reference.

Chapters 3 through 10 provide more detail on topics introduced in Chapters 1 and 2:

- [Chapter 3, “Using Workspaces and Jobs,”](#) discusses how to use workspaces and jobs.
- [Chapter 4, “Defining the Data,”](#) discusses how to introduce data into a job.
- [Chapter 5, “Creating Data Flow Diagrams,”](#) discusses how to create data flow diagrams within jobs.
- [Chapter 6, “Selecting and Using Operators,”](#) discusses how to use operators within data flow diagrams.
- [Chapter 7, “Working with Snippets,”](#) discusses how to use visual snippets within operators.
- [Chapter 8, “Creating and Modifying Flow Engine Machines,”](#) discusses how to configure the job environment.
- [Chapter 9, “Generating and Running a Job,”](#) discusses how to generate a job from a data flow diagram and run that job on the target platform.
- [Chapter 10, “Advanced Job Topics,”](#) discusses advanced job topics.

[Appendix A, “DocSample2: An Extended Example of a Data Flow Diagram,”](#) discusses a more complex data flow diagram that builds on the data flow diagram introduced in Chapter 2.

[Appendix B, “Summary of Supported Data Types,”](#) discusses the datatypes supported by the Informix Formation product.

[“Glossary”](#) contains definitions for the key terms used in this guide.

Related Documentation

The Informix Formation documentation set includes the following documents.

<i>Informix Formation Installation and Configuration Guide</i>	This guide. Describes how to install and configure Formation on UNIX or Windows NT systems. This guide is available in print.
<i>Informix Formation User's Guide</i>	Describes how to use the Formation product. This guide is available in print.
Informix Formation Architect Online Help	Describes how to use the Formation Architect graphic user interface (GUI) components, including menu items, operators, and visual snippets. The Architect online help is installed as part of Formation as a Microsoft Windows Help (WinHelp) file.
Release Notes	Describes information that was unavailable when the other information was produced. The release notes are available online as the file readme.txt

In addition to the Formation documentation set, the following documents may also be useful:

- Documentation for the one or more databases you use with Formation, such as the Informix Dynamic Server documentation set
- Documentation for your hardware and operating system platforms

Conventions

This manual uses the following textual conventions.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
<i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
<code>monospace</code>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.

Syntax Notation

This guide uses the following conventions to describe the syntax of operating system commands.



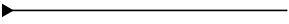
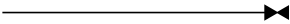


Command Element	Example	Convention
Values and parameters	<i>table_name</i>	Items that you replace with an appropriate name, value, or expression are in <i>italic</i> type style.
Optional items	[]	Optional items are enclosed by square brackets. Do not type the brackets.
Choices	ONE TWO	Choices are separated by vertical lines; choose one if desired.
Required choices	{ONE TWO}	Required choices are enclosed in braces; choose one. Do not type the braces.

Command Element	Example	Convention
Default values	<u>ONE</u> TWO	Default values are underlined, except in graphics where they are in bold type style.
Repeating items	name, ...	Items that can be repeated are followed by a comma and an ellipsis. Separate the items with commas.
Language elements	() , ; .	Parentheses, commas, semicolons, and periods are language elements. Use them exactly as shown.

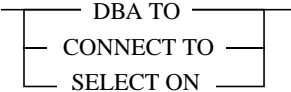
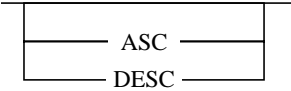
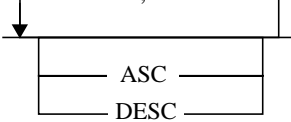
(2 of 2)

Syntax Diagrams

This guide uses diagrams built with the following components to present the syntax for statements and all commands other than system-level commands.

Component	Meaning
	Statement begins.
	Statement syntax continues on next line. Syntax elements other than complete statements end with this symbol.
	Statement continues from previous line. Syntax elements other than complete statements begin with this symbol.
	Statement ends.
	Required item in statement.
	Optional item.

(1 of 2)

Component	Meaning
	Required item with choice. One and only one item must be present.
	Optional item with choice. If a default value exists, it is printed in bold.
	Optional items. Several items are allowed; a comma must precede each repetition.

(2 of 2)

Keywords and Punctuation

Keywords are words reserved for statements and all commands except system-level commands. When a keyword appears in a syntax diagram, it is shown in uppercase. You can write a keyword in upper- or lowercase, but you must spell the keyword exactly as it appears in the syntax diagram.

Any punctuation that occurs in a syntax diagram must also be included in your statements and commands exactly as shown in the diagram.

Identifiers and Names

Metavariables serve as placeholders for identifiers and names in the syntax diagrams and examples. A metavariable can be replaced by an arbitrary name, identifier, or literal, depending on the context. Metavariables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a metavariable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses metavariables to illustrate the general form of a simple SELECT statement:

►— SELECT — *column_name* — FROM — *table_name* —►

When you write a SELECT statement of this form, you replace the metavariables *column_name* and *table_name* with the name of a specific column and table.

Introduction to Informix Formation

Introduction to Informix Formation	1-3
Overview of the Formation Components	1-4
The Formation Architect	1-7
The Formation Architect User Interface	1-8
The Data Flow Diagram	1-9
The Formation Flow Engine.	1-11

Introduction to Informix Formation

Informix Formation™ is a data transformation tool that substantially reduces the time and complexity needed to construct and update a data warehouse. Formation does this by providing two components:

- An intuitive graphical user interface, the *Formation Architect*™, that allows you to visually define the data transformations to be performed.
- A powerful execution engine, the *Formation Flow Engine*™, that makes use of a parallelized data flow execution model to speed the processing of large complex data transformations.

Through these components, Formation provides the scalability and performance required to cope with the increasingly large volume and complexity of business data available from a variety of sources, such as databases, operational systems, third-party syndication data, and web site activity logs.

This chapter contains the following sections that introduce the Formation software and provide an overview of the product features and functionality:

- [Overview of the Formation Components](#)
- [The Formation Architect](#)
- [The Formation Flow Engine](#)

Overview of the Formation Components

The Formation product consists of the following components:

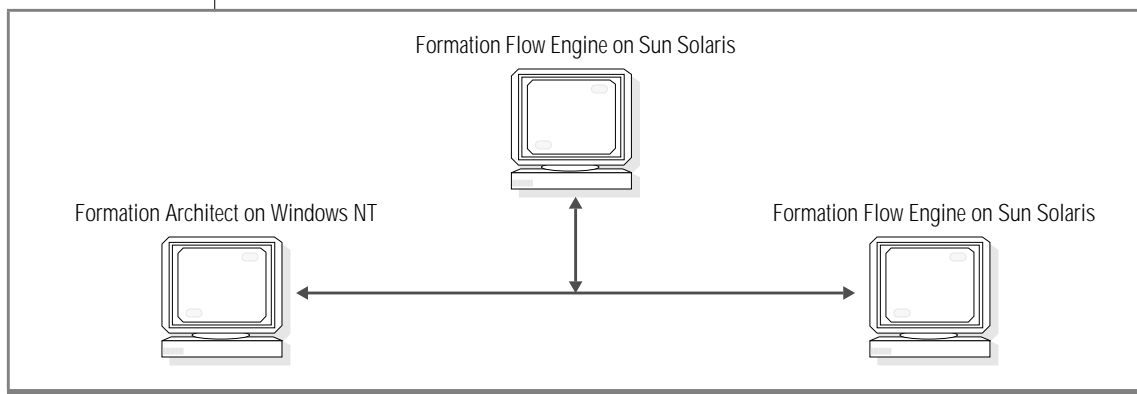
- **Formation Architect™**

The graphical tool used to design, evolve, and generate the data integration jobs. The Formation Architect runs on the Windows NT platform. For more information on the Formation Architect, refer to [“The Formation Architect” on page 1-7](#).

- **Formation Flow Engine™**

The software that executes a Formation-generated data transformation job. The Formation Flow Engine runs on either UNIX or Windows NT systems. For more information on the Formation Flow Engine, refer to [“The Formation Flow Engine” on page 1-11](#).

The Architect and the Flow Engine can be installed in a stand-alone configuration with both components installed on the same Windows NT system, or they can be installed in a distributed configuration, with the Flow Engine installed on a remote UNIX system. The following figure shows a distributed installation with a single Architect working with remote Flow Engines on Sun Solaris systems.



The process of defining, generating, and executing a data transformation job using Formation is a straightforward process. The following are the steps involved in this process:

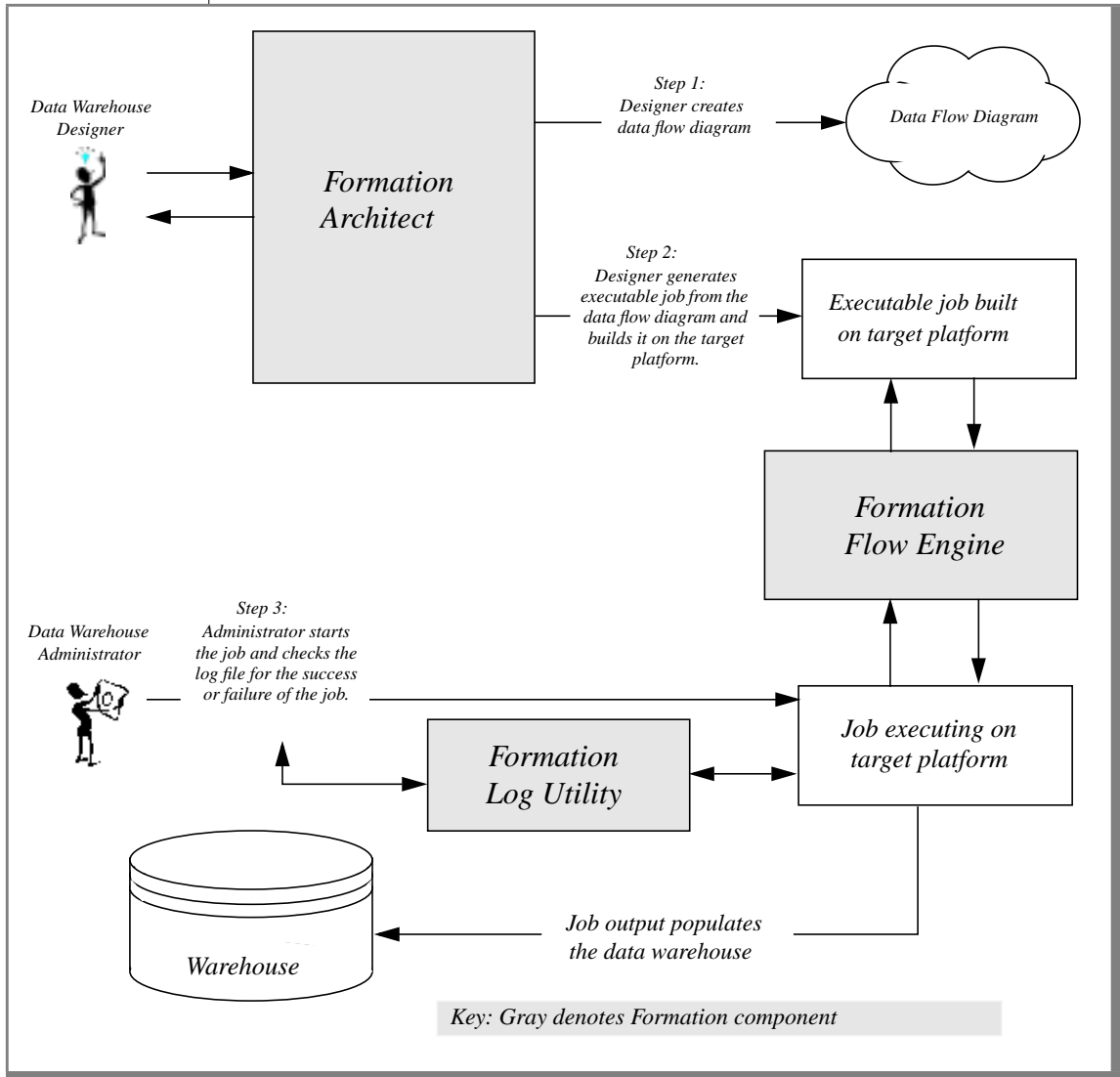
1. Use the Architect to create a data flow diagram that defines the source data, the target data, and the transformation processes that must occur from source to target. For more information on constructing data flow diagrams, refer to [Chapter 5, “Creating Data Flow Diagrams.”](#)
2. When the data flow diagram is complete, generate C++ source files that mirror the transformations specified in the data flow diagram. Along with the source files, the program generates a runtime parameters file that stores information about the job configuration.

As part of the code generation process, Formation copies the source files to the Flow Engine server, where they are compiled and linked to create executable programs.
3. The final step is to run the executable files against the source data, which actually transforms the data and then loads it into the target location, usually a data warehouse. While the job executes, status messages are written to the Formation Log Utility so that there is a step-by-step record of the actions taken during the job.

For more information on generating and running a job, refer to [Chapter 9, “Generating and Running a Job.”](#)

Overview of the Formation Components

The following illustration shows the steps in generating a data warehouse using Formation.



The Formation Architect

You use the Architect to create a visual representation of the steps needed to extract source data, transform it, and then load it into a data warehouse.

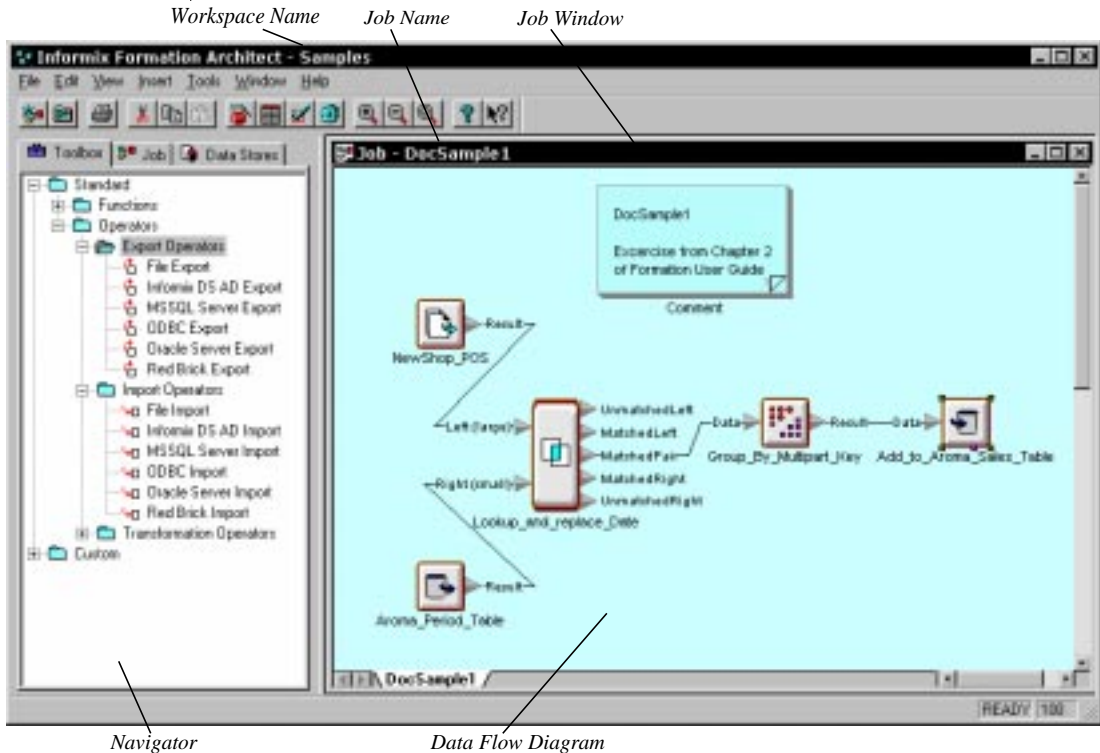
The Architect offers a “drag and drop” interface that enables you to quickly assemble a complex job, then drill down to specify levels of complexity for each operator in the job. For example, you can drag the Join operator into the job window, then use the MatchedPair Project parameter to visually map data coming into the join to the desired format of the data exiting the join.

After you specify the work the job must perform, you can generate C++ source code, move it to the Flow Engine server, and compile and link it, all in one step.

The Formation Architect User Interface

The main window of the Formation Architect appears as follows.

Figure 1-1
The Formation Architect



The Formation Architect (shown on the previous page) contains several important elements, as follows:

- The *Navigator* enables you to navigate the contents of a workspace. All jobs and data definitions are stored in workspaces. The Navigator has the following tabs:
 - The *Toolbox* tab displays the standard operators and functions provided with the software, as well as any operator variables and custom functions you have created while working in the active workspace.
 - The *Job* tab shows the graphical pieces of the active job.
 - The *Data Stores* tab contains the data definitions defined in the active workspace.
- The *Workspace Name* lists the active workspace.
- The *Job Window* displays the operators and data flows in a job. The title bar of the job window shows the name of the job. Inside the job window, the *data flow diagram* is the graphical representation of the data transformations that will occur in the job.

For more information on using the Architect graphical user interface (GUI), refer to the Architect online help.

The Data Flow Diagram

The data flow diagram is the visual representation of the data transformations that you want the job to execute on the source data.

You construct a data flow diagram by dragging and dropping operators into the job window and connecting them together. The following graphic shows a typical data flow diagram.

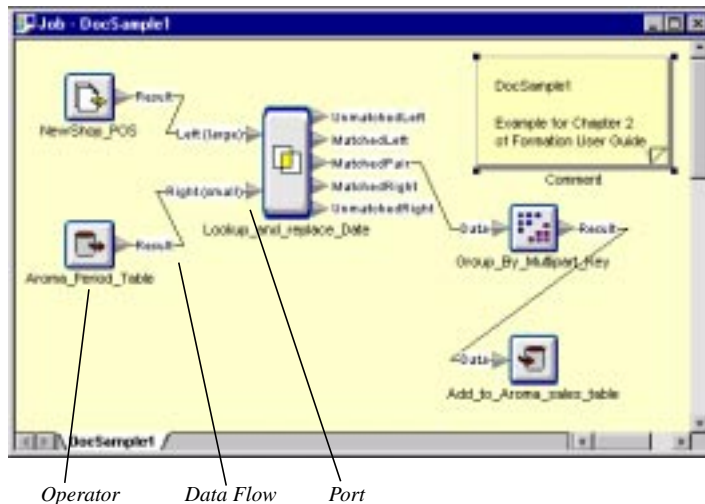


Figure 1-2
A Data Flow Diagram

The *data flow* is the processing path that data takes from one operator to another. A data flow is represented in a data flow diagram as a line connecting two operators at their ports. In this example, the data flow indicated represents the data path between the Red Brick Import operator named `Aroma_period_table` and the Join operator named `Lookup_and_replace_Date`. For more information about data flow diagrams, refer to [Chapter 5, “Creating Data Flow Diagrams.”](#)

An *operator* represents some data extraction or transformation activity, such as sorting or filtering data. In this example, the operator indicated is a Red Brick Import operator named `Aroma_period_table`.

A *port* is the part of an operator at which a data flow is connected. In this example, the port indicated is an input port to the Join operator and the port is named `Right(small)`. For more information about operators, refer to [Chapter 6, “Selecting and Using Operators.”](#)

The Formation Flow Engine

The Flow Engine coordinates the execution of jobs by providing an environment in which a single job executes as multiple processes.

The Flow Engine is installed on the platforms on which you process your data. The Flow Engine uses a multiple input, multiple output, multiple operation parallel execution data model. The Flow Engine has a batch parallel execution data model where you can have one or more flows of data on an SMP computer, each executing different parts of the data integration job.

For example, a single input data flow can be split into several data flows using the Partition operator in the data flow diagram for that job. Some or all of those data flows could then be combined using the Gather operator. This execution model gives Formation speed and flexibility, since the multiple data flows and processing take advantage of multiprocessing hardware and software.

Further, a data transformation process can take a set of input data sources and produce a different set of output data targets. This capability is an important aspect of Formation performance because it allows the number of passes over source data to be reduced, the need for storing intermediate results to be minimized, and intermediate results to be used simultaneously in several branches of a data flow diagram.

For example, to produce ten tables with overlapping data members from a single data set using SQL, you need to process that data set ten times. Using the Formation product, you can produce those ten tables of data in a single pass, by directing the overlapping data members to more than one data flow.

DocSample1: A Quick Start Guide

In This Chapter	2-3
Opening the DocSample1 Job	2-3
Analyzing the DocSample1 Job.	2-5
Defining the Data Stores.	2-6
The NewShop Data Store	2-7
The Period Data Store	2-8
The Sales Data Store.	2-8
Setting Up the Import and Export Operators	2-9
The NewShop_POS File Import Operator	2-10
The Aroma_Period_Table Red Brick Import Operator	2-10
The Add_to_Aroma_sales_table Red Brick Export Operator	2-11
Joining the Data Flows	2-11
Defining the Input Data Definitions	2-12
Defining the Output Data Definition	2-12
Selecting Keys	2-13
Projecting the Input Data to the Output Data Definition	2-14
Grouping the Data.	2-15
Selecting the Keys	2-16
Defining the Summary Record	2-16
Initializing the Summary Record	2-17
Mapping the Input to the Summary Record	2-18
Calculating the Summaries	2-18
Finalizing the Output Record	2-19
Exporting the Data	2-20
Generating and Building the DocSample1 Job	2-21
Selecting Generation Options	2-22
Running and Monitoring the DocSample1 Job	2-23
Looking at the Executable Files	2-23

Starting the Job 2-23

Viewing the Job Log 2-24

In This Chapter

This chapter describes the process of creating and running a sample job as a way to familiarize yourself with the Informix Formation product. Topics introduced in this chapter are discussed in detail in later chapters of this book.

The chapter is organized into the following sections:

- [Opening the DocSample1 Job](#)
- [Analyzing the DocSample1 Job](#)
- [Generating and Building the DocSample1 Job](#)
- [Running and Monitoring the DocSample1 Job](#)

This sample job uses two tables from the Aroma database, which is part of the Red Brick Warehouse product. Successfully executing this sample job requires access to the Aroma database.

For more information on the operators used in the data flow diagram in this job, refer to [Chapter 6, “Selecting and Using Operators,”](#) or the Architect online help.

For detailed information on creating data flow diagrams, refer to [Chapter 5, “Creating Data Flow Diagrams.”](#)

Opening the DocSample1 Job

This sample job, named DocSample1, takes data from a fictional store and runs it through two sets of transformations. These transformations change the data format and the data itself so that it is ready to be added to the Aroma sample database shipped with the Red Brick Warehouse product.

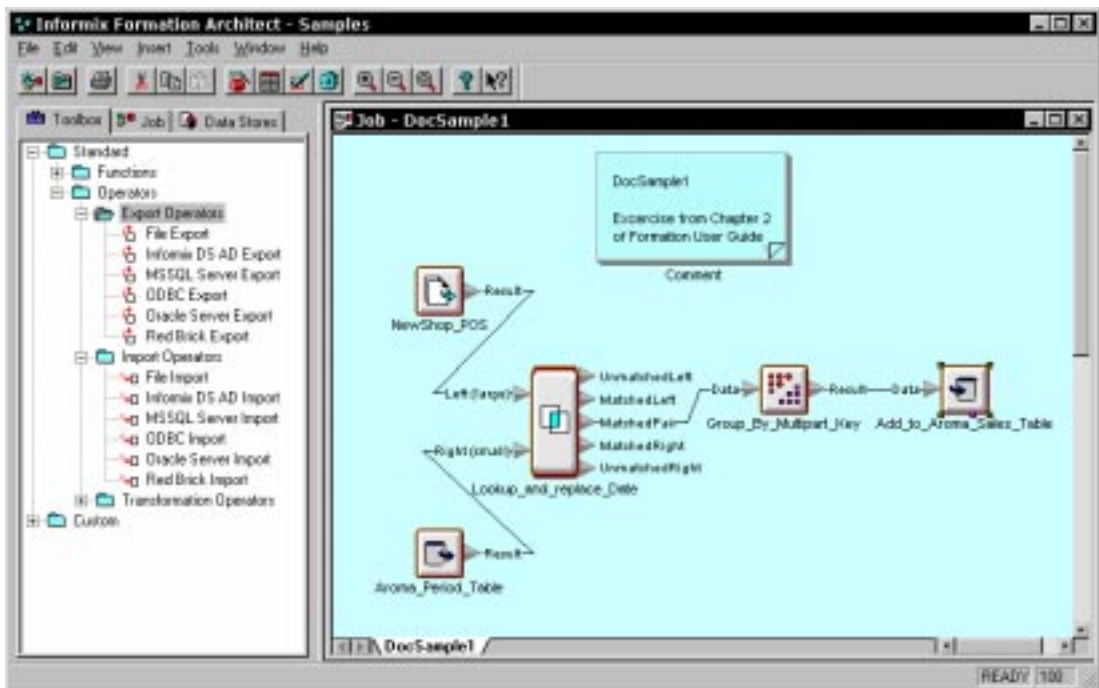
Opening the DocSample1 Job

To open the sample job:

1. Select Informix Formation Architect from the Windows NT Start menu to start the software. The Manage Jobs dialog is displayed.
2. Select the DocSample1 job from the Samples workspace and click the Open button.

The DocSample1 job is displayed in the job window as shown in the following illustration.

Figure 2-1
The DocSample1 Job



Analyzing the DocSample1 Job

The DocSample1 job is a response to a retail store chain's problem. The store owner must take data from a store, called NewShop, change the date format to match a standard key used in the corporate data warehouse, and then add the transformed data into the Sales table in the Aroma data warehouse.

The following illustration shows the data flow diagram for DocSample1.

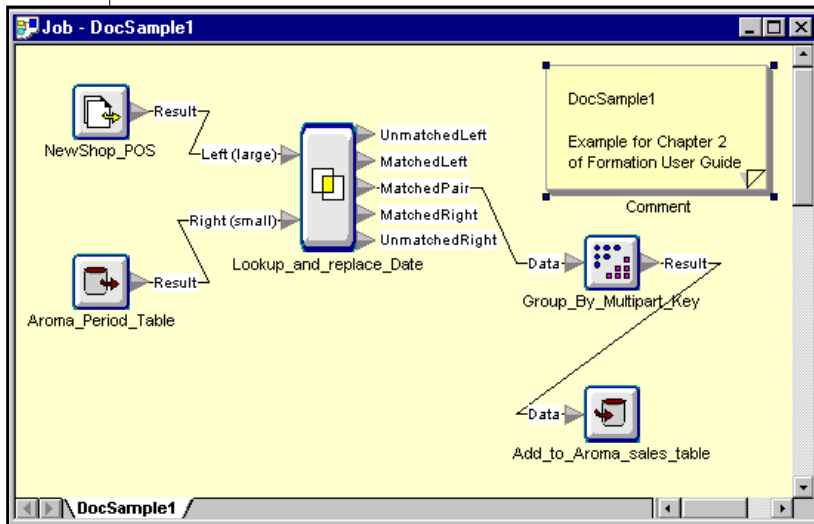


Figure 2-2
DocSample1 Data
Flow Diagram

The rest of this section analyzes the job creation using the following methodology:

- Defining the Data Stores
- Setting Up the Import and Export Operators
- Joining the Data Flows
- Grouping the Data
- Exporting the Data

Defining the Data Stores

The first step in creating the job is to define the structure of the source data and of the target data. These definitions are called *data stores*, and they appear on the Data Stores tab of the Navigator. There are two types of data stores:

- Database schemas, which can contain one or more table definitions from an existing database.

The Data Definition Wizard in the Formation Architect uses existing ODBC data stores to automatically import the schemas from one or more tables in the following databases:

- ☐ Informix Dynamic Server
 - ☐ Informix Dynamic Server/AD
 - ☐ Red Brick Warehouse
 - ☐ Oracle Server 7 or Oracle Server 8
 - ☐ Microsoft SQL Server
- File schemas, which define fields for text files or flat files.
For files, you step through the Data Definition Wizard to manually define the file structure.

The DocSample1 job requires the following three data stores:

- The NewShop data store for the store's data
- The Period data store for the date and key mapping
- The Sales table where the transformed data will be stored

The NewShop Data Store

The NewShop data store is a text file data store that defines the structure of the data generated by the Point-Of-Sale (POS) system used by the store named NewShop. This data store contains the following fields:

Field	Description	Data Type
Date	Date formatted as YYYY-MM-DD.	Date
ClassKey	First part of two-part key that identifies each product.	Integer
ProductKey	Second part of two-part key that identifies each product.	Integer
StoreKey	Identifies this store.	Integer
PromoCode	Code for each item on sale.	Integer
Quantity	Number of items sold.	Integer
Amount	Dollar amount of sale.	Decimal, precision of 2

The file that stores the NewShop data is located in the Samples directory under the main Formation directory. This file is located in the Samples subdirectory of your Formation Architect installation and is named newshoppoos.txt.

The Period Data Store

The Period table is an Aroma database table that links a specific date to an integer key. The date in the NewShop input file must be replaced with the period key before the data can be added to the Sales table. The Period table has the following structure:

Field	Description
perkey	Integer that identifies exactly one row in the Period table.
date	Date value that identifies each day in the format YYYY-MM-DD.
day	Character-string abbreviation of the day of the week. Examples: SA, SU, MO.
week	Integer that identifies each week of each year by number, with each week starting on a Sunday.
month	Character-string abbreviation of the name of each month. Examples: JAN, FEB, MAR.
qtr	Character string that uniquely identifies each quarter. Examples: Q1_94, Q3_95.
year	Integer that identifies the year. Examples: 1994, 1995, 1996.

The Period table is part of the Aroma sample database, which is shipped with the Red Brick Warehouse product.

The Sales Data Store

The Sales table is an Aroma database table that stores summary data about sales based on a five-part key: period, class, product, store, and promotion code. The Sales data store has the following structure:

Field	Description
perkey	Key to the Period Table (one key for each unique date)
classkey	Key to Class Table
prodkey	Key to Product Table

Field	Description
storekey	Key to Store Table
promokey	Key to Promotion Table
quantity	Amount of sales per day
dollars	Dollars per day.

(2 of 2)

The Sales table is part of the Aroma sample database, which is shipped with the Red Brick Warehouse product.

Setting Up the Import and Export Operators

The next step in the job creation is to specify sources and targets.

The data flow diagram begins by bringing data into the diagram using one or more import operators. This data flow diagram uses two different import operators, as described in the following table.

Name in Data Flow Diagram	Operator and Description
NewShop_POS	File Import Reads the data from the <i>newshoppo.txt</i> file and creates a data flow containing all records in that text file.
Aroma_Period_Table	Red Brick Import Reads records from the Period table in the Aroma Red Brick Warehouse database to the input data flow.
Add_to_Aroma_sales_table	Red Brick Export Writes data from the data flow to the Sales table in the Aroma Red Brick Warehouse database.

For more information on these operators, refer to [Chapter 6, “Selecting and Using Operators.”](#)

The NewShop_POS File Import Operator

The File Import operator, NewShop_POS, is set up to read sales data for the store from the file newshoppo.txt.

Parameter Values

In this example, the required parameters you must specify are set to the following values:

Record Definition: NewShopPOS data store

File List: newshoppo.txt in the Formation Samples directory.

Important: You may need to edit this location if you installed the Formation Architect in a location other than the default (C:\Program Files\Informix\Formation).



The Aroma_Period_Table Red Brick Import Operator

The Red Brick Import operator, Aroma_period_table, is set up to read data from the Period table in the Aroma database. The required parameters you must specify for this operator are set to the following values:

Parameter Values

Table Definition: Period data store

Database: aroma

Table Name: period

User Name: system

Password: manager

The Add_to_Aroma_sales_table Red Brick Export Operator

The Red Brick Export operator, Add_to_Aroma_sales_table, is set up to write the data at the end of the data flow to the Sales table in the Aroma database. The required parameters you must specify are set to the following values:

Parameter Values

Table Definition: Sales data store
Table Name: sales
User Name: system
Password: manager
Load mode: modify

Joining the Data Flows

The next step is to join the two data flows together and use the Project parameter in the Join operator to redefine the output by dropping some columns from both input data flows. The data flow diagram uses the Join operator, described in the following table, to perform these tasks.

Name in Data Flow Diagram	Operator and Description
Lookup_and_replace_Date	Join operator. In this example, the join creates one table of matched records and another table for shop records that did not match items in the period table.

At this point in the data flow diagram, each record in the NewShop data flow contains a date of the transaction. The purpose of the Join is to replace each real date with a period key that maps to that date, because the Sales table in Aroma requires the period key and not the actual date.

Defining the Input Data Definitions

To begin the setup of the Join operator, the two data flows are connected to the two input ports of the operator as follows:

- The data flow from the NewShop_POS import operator output port is connected to the Left (large) input port. The data flow with the larger number of records is usually attached to this port.
- The data flow from the Aroma_Period_table import operator is connected to the Right (small) input port. The smaller data flow is usually attached to this port.

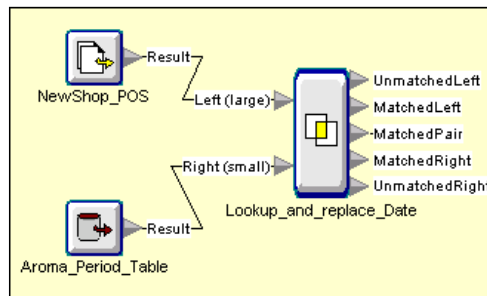
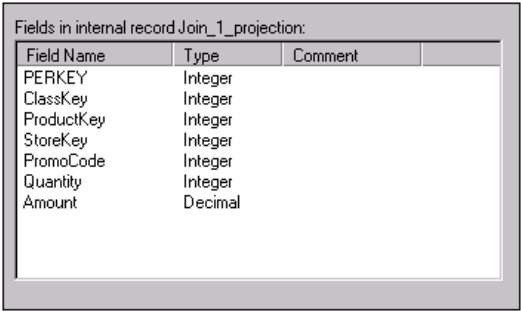


Figure 2-3
Defining the Input Data

Defining the Output Data Definition

By default, the MatchedPair output port on the Join operator uses a data definition that contains all the fields in both input data flows. In this job, the default data definition on the output port contains all fields from the Period table and all fields from the NewShopPOS file.

Because the purpose of the Join operator is to look up a period key for the Date value and replace it, the data definition for the records coming out of the Join operator does not need to include the Date field from the NewShopPOS data flow or most of the fields in the Period table. Therefore, all fields from the Period table except PERKEY were deleted and the Date field from the NewShop file was also deleted. Then the PERKEY field was then moved up to the top of the list.



Field Name	Type	Comment
PERKEY	Integer	
ClassKey	Integer	
ProductKey	Integer	
StoreKey	Integer	
PromoCode	Integer	
Quantity	Integer	
Amount	Decimal	

Figure 2-4
*Moving the PERKEY
field to the top of the
list*

Selecting Keys

The Join operator requires a user-specified key for each table that is used to match the records, so the job must specify Date as the key for both the Left key (the store data, that contains the date of each transaction record) and the Right key (where the date is part of a record that contains a Period key required by the Sales table).

Projecting the Input Data to the Output Data Definition

When the two keys match, the matching records are combined, the date fields are dropped from the combined record, and the resulting record is sent to the MatchedPair output port. Because the output data definition uses some, but not all, columns from both input data definitions, you have to specify which input columns map to which output columns. This is the purpose of the MatchedPair Project snippet parameter of the Join operator.

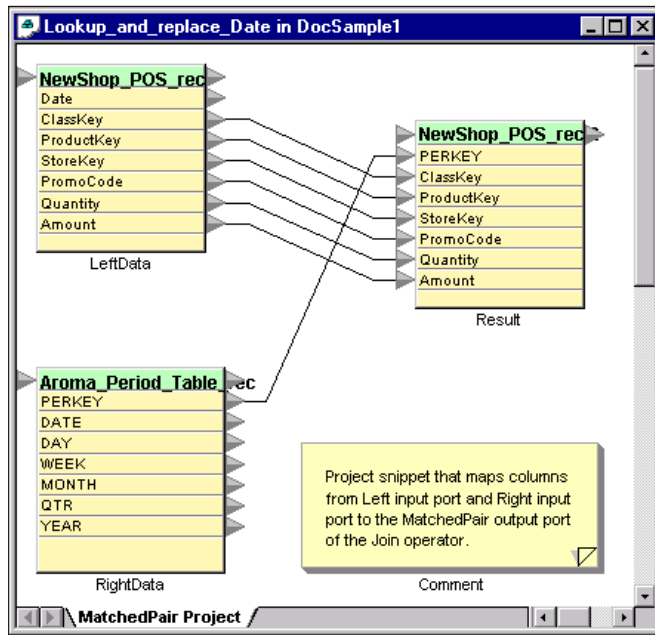


Figure 2-5
The MatchedPair
Project Snippet

The data blocks on the left side of the snippet window (LeftData and RightData) represent the two input data flows. The data block on the right side (Result) represents the data flow coming out of the join operator. The lines from left to right show the fields from the input data blocks that have been mapped to fields in the output data block.

Grouping the Data

After the two input flows have been joined, the data needs one more transformation before it is ready to add to the Sales table. The quantity and amount of sales must be grouped by a five-part key and then totalled to produce the total quantity and total sales by key.

Name in Data Flow Diagram	Operator and Description
Group_By_Multipart_Key	Group By Collects the input data flow by one or more key fields, aggregates the data by the one or more key fields, and writes the summarized data to the output data flow.

To start the grouping, the data flow from the MatchedPair output port of the Join operator is connected to the Data input port on the Group By operator.

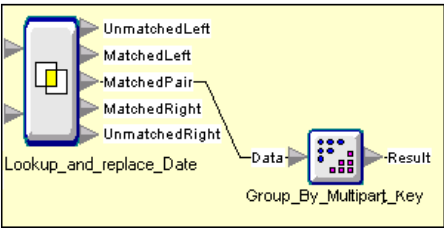


Figure 2-6
Connecting the MatchedPair Port to the Group By Operator

The Group By operator setup is explained in the following sections.

Selecting the Keys

The Sales table stores the sales data using a multi-part key: period, class, product, store, and promotion code. So all five fields are selected in that order as the key to use in grouping the records.

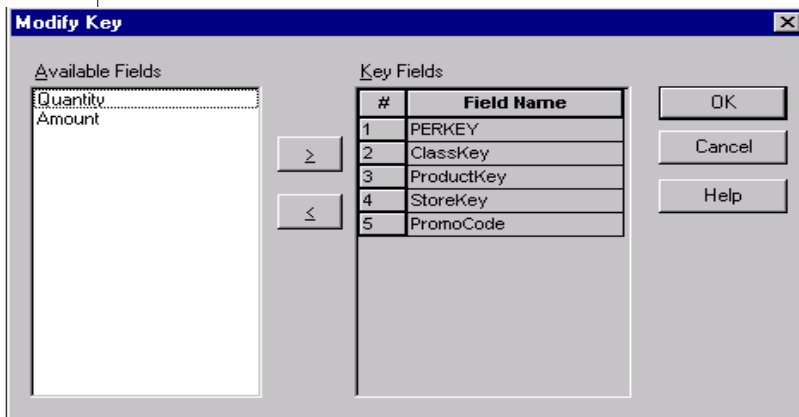


Figure 2-7
Defining the Key

Defining the Summary Record

Next, the fields to be aggregated are specified as the summary record. The Quantity field (an integer) and the Amount field (decimal, with a precision of 2) are the two fields in this summary record. Think of the summary record as a running total of the calculations done for each matching group of records by the Group By operator.

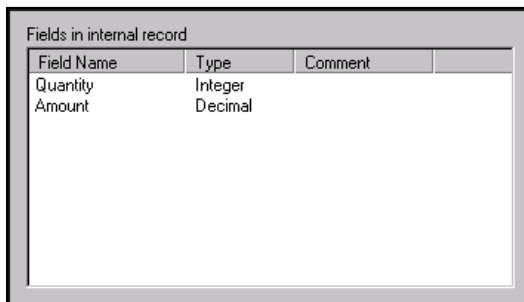


Figure 2-8
Fields in the
Summary Record

Initializing the Summary Record

In this job, each time the Group By operator finds a new key, all fields in the summary record are initialized to zero in the Initialize snippet parameter.

Because the Formation Architect supports automatic data conversion whenever possible, a single integer value of 0 (zero) is converted to 0.0 when connected to the decimal Amount field; the circle on the line between the constant and the Amount field denotes an automatic conversion.

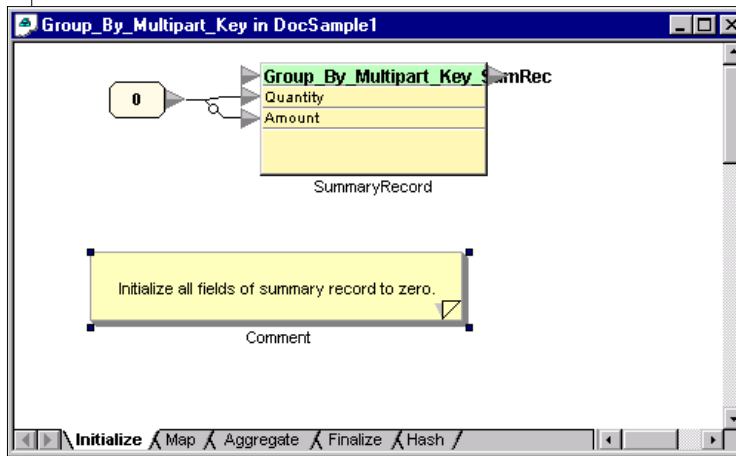


Figure 2-9
The Initialize
Snippet

Mapping the Input to the Summary Record

The Map snippet parameter maps the fields in the input data definition to the fields defined in the summary record. In the sample, there is a simple one-to-one mapping between the Quantity and Amount fields, indicated by the lines connecting the fields in the Data block to fields in the summary record.

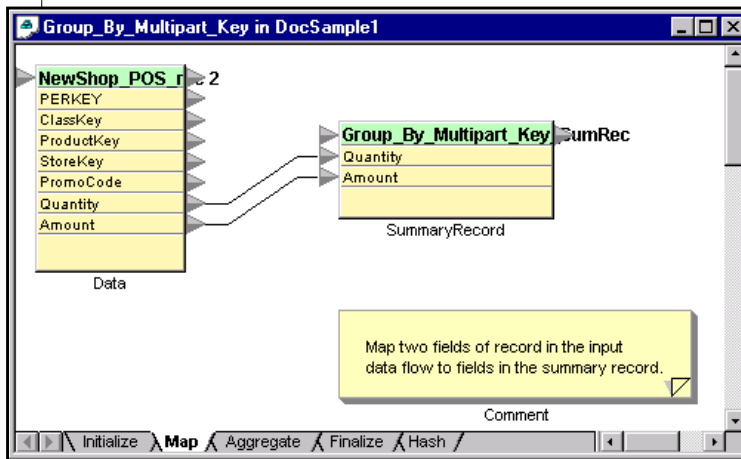


Figure 2-10
The Map Snippet

Calculating the Summaries

The Aggregate snippet parameter is where calculations take place.

The two SummaryRecord blocks represent temporary storage for all the records that are being aggregated by the Group By operator. To produce the final, grouped record for the entire group of records, you combine the two SummaryRecord blocks into the SummaryRecordResult block.

In this job, the Quantity fields are added together and stored in the Quantity field of the SummaryRecordResult block. In the same fashion, the Amount fields are added together and stored in the Amount field of the SummaryRecord Result block.

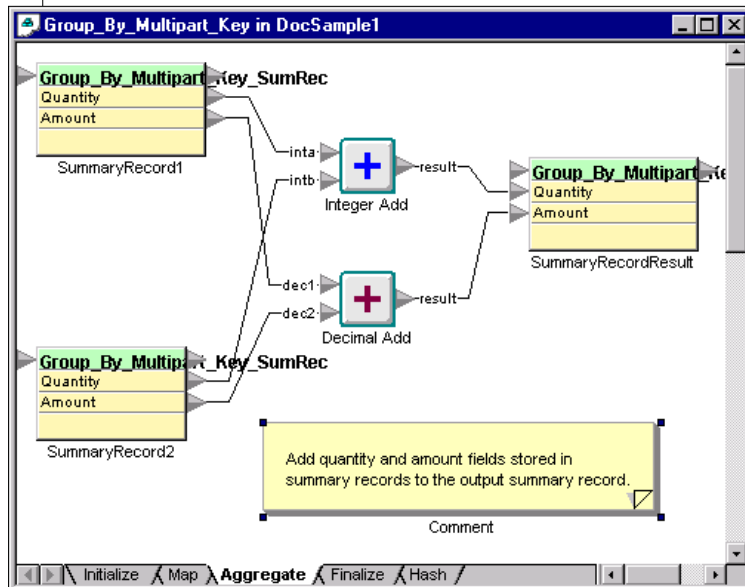


Figure 2-11
The Aggregate Snippet

Finalizing the Output Record

When all the records with a specific key have been processed, the final summary record is ready to be written to the output record.

To create the Result block for this snippet, the data flow from the input port of the next operator, Red Brick Export, is connected to the output port of the Group By operator. This action automatically propagates the data definition from the export operator to the Group By, ensuring that the data flow produced by the Group By operator matches the format of the data flow required by the export operator.

The Finalize snippet parameter maps the fields in the summary record to fields in the output record, and the fields used as keys are also mapped to output record fields.

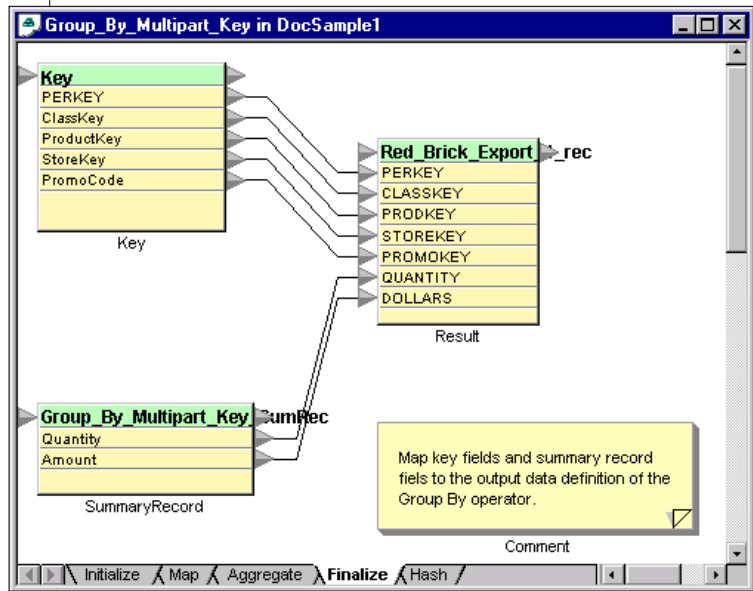


Figure 2-12
The Finalize Snippet

Exporting the Data

Now that the data matches the exact record format required by the Aroma Sales table, the data flow diagram concludes with the data flow being directed into a Red Brick Export operator, which will add the data to the Sales table.

Name in Data Flow Diagram	Operator and Description
Add_to_Aroma_sales_table	Red Brick Export Writes data from the data flow to a Red Brick Warehouse database table.

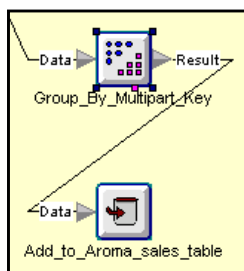


Figure 2-13
Connecting the Data
Flow to the Red
Brick Export
Operator

Because the parameters to the Red Brick Export operator were defined in the second stage of the job creation, all you have to do to complete the data flow diagram is to connect the data flow from the Group By operator to the Red Brick Export operator.

Generating and Building the DocSample1 Job

Once you have designed a job, the next step is to generate the source code files and then build the executable files with the Formation Flow Engine.

Important: This section assumes that the job files will be generated and built on the same Windows NT computer where the Formation Architect is installed. This PC must have both the Formation Flow Engine and Microsoft Visual C++ Version 6.0 installed before building the executable files and running the job.



Selecting Generation Options

To begin the process of generating source files and building executable files, you select the options you want to use in the Generate Job Files dialog shown in the following illustration.

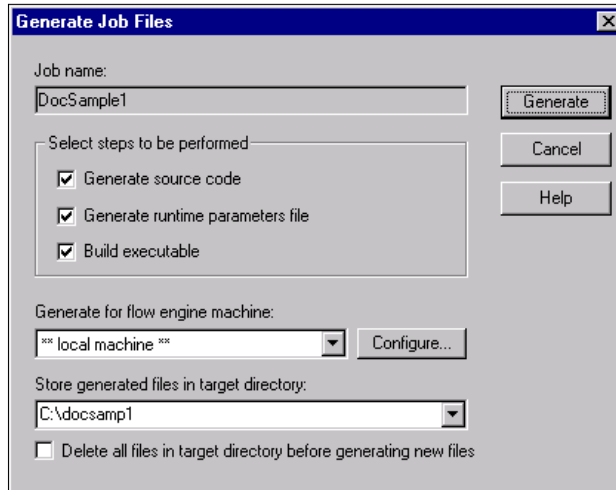


Figure 2-14
*Generating and
Building the
Executable Files*

To select the options:

1. Select Generate Job Files from the Tools menu. The Generate Job Files dialog is displayed.
2. Make sure all three check boxes are selected:
 - Generate source code
 - Generate runtime parameters file
 - Build executable
3. Select a location where the generated files will be stored. Be sure to place the files for each job in a separate directory.

If the directory you specify does not exist, the operating system will create it when the job is run.

4. Click the Generate button to generate the job files and use those files to build the job executables.

The Build Status dialog is displayed, and lists the copy, generate and build steps performed. When the build is finished, the OK button is activated.

5. Click the OK button to exit the Build Status dialog.

For more information on generating and building a job, refer to [Chapter 9, “Generating and Running a Job.”](#)



Running and Monitoring the DocSample1 Job

Once you have built the job executable files, the final step is to run the job on the actual data. You run the job executable files from the operating system prompt, outside of the context of the Formation Architect component.

***Tip:** In this example, the job files are built and run on the Windows NT computer on which the Formation Flow Engine is installed.*

Looking at the Executable Files

The build process compiles and links the files so that they are ready for execution. Usually, more than one executable file is created, so the Formation Flow Engine uses a master executable, `rbf_job`, that schedules and tracks the running of the other executable files in the job.

Starting the Job

To start running the job, you do the following:

1. Open an MS-DOS window.

2. In the MS-DOS window, run the master executable job file, `rbf_job jobname`, where *jobname* is an optional name you specify, such as `MyJobRun3`. This job name is used in the log file to identify messages about the job. Specify a job name of `MyJobRun3` as follows:

```
rbf_job MyJobRun3
```

This `rbf_job` file executes the other executable portions of the job in the correct order. When the job finishes, the MS-DOS prompt is displayed.

3. You can now check the log file for the job status and, if it ran successfully, you can query the Aroma Sales table to view the newly appended data.

Viewing the Job Log

Use the Formation Log Utility to view the log file of the processing of the job executables.

To use the Log Utility on the Windows NT platform, open an MS-DOS window and run the Log Viewer, `rbflogview`, as follows to view the results of the execution of the job `MyJobRun3`:

```
rbflogview -a -i MyJobRun3 > run3log.txt
```

Using the previous command causes all log entries in the currently active log file for the job run named `MyJobRun3` to be written to a file named `run3log.txt`. For more information on using the Formation Log Utility, refer to [Chapter 9, “Generating and Running a Job.”](#)

Using Workspaces and Jobs

In This Chapter	3-3
Using Workspaces	3-3
Creating, Copying, or Importing a Workspace	3-4
Opening a Workspace	3-7
Exporting a Workspace for Backup	3-7
Renaming and Deleting Workspaces	3-9
Using Jobs	3-10

In This Chapter

This chapter describes how to use workspaces and jobs in the Informix Formation product. A workspace contains one or more jobs.

For an overview of the entire process of using the Formation product, [Chapter 2, “DocSample1: A Quick Start Guide.”](#)

This chapter is organized into the following sections:

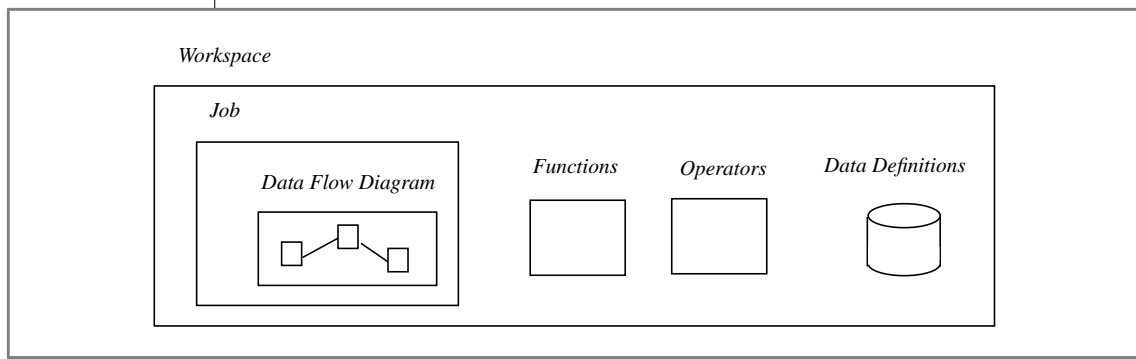
- [Using Workspaces](#)
- [Using Jobs](#)

Using Workspaces

Any work you do with the Architect tool is done in the context of a job or a workspace. For example, the DocSample1 and DocSample2 jobs are contained in the Samples workspace.

A workspace is a named collection of data definitions, operators, functions, and jobs that you use to do your work. A job is a named container of a data flow diagram. The data flow diagram graphically describes the data transformation process by combining operators, functions, and data definitions from the workspace.

The following figure illustrates this relationship.



You use workspaces to organize, backup, and share your work. For example, you may have one workspace for each project, and you may routinely backup these workspaces for safekeeping or for sharing with coworkers.

The following sections describe how to perform common workspace tasks:

- [Creating, Copying, or Importing a Workspace](#)
- [Opening a Workspace](#)
- [Exporting a Workspace for Backup](#)
- [Renaming and Deleting Workspaces](#)

Creating, Copying, or Importing a Workspace

The first few times you use the Architect, you use the default Samples workspace to familiarize yourself with the Formation software.

When you are ready to create your own jobs, you should create new workspaces to contain them rather than using the Samples workspace. This preserves the Samples workspace for future reference. In addition, the Samples workspace is a part of the Formation product, and will be replaced when you install a newer version of the Formation product. This replacement would result in the loss of any jobs, data stores, or custom functions you had placed in the old Samples workspace.

To create a new workspace, use the Formation Workspace Manager tool. You start the Workspace Manager by selecting Workspace Manager from the Informix Formation folder on the Start menu on a Windows NT system. You cannot start the Workspace Manager tool when the Architect tool is already executing. Exit the Architect before starting the Workspace Manager tool.

The Workspace Manager appears as in the following illustration.

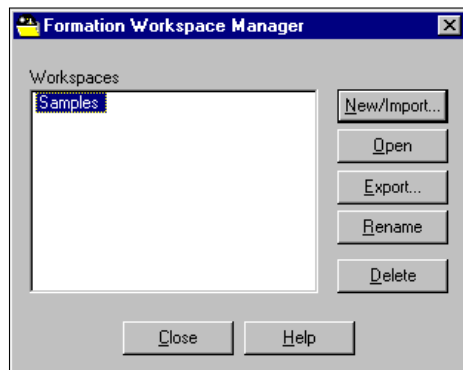


Figure 3-1
The Workspace Manager Dialog

In the previous illustration, you see that only the Samples workspace is defined, since it is the only workspace displayed in the Workspaces box.

To add a new workspace, click the New/Import button. The New/Import Workspace dialog is displayed as shown in the following illustration.

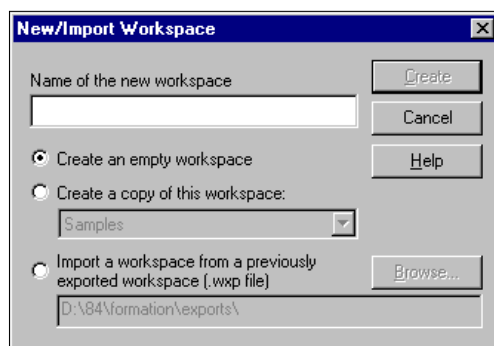


Figure 3-2
The New/Import Workspace dialog

Enter the name of the new workspace, and then select the type of workspace you want to create:

- Create a new empty workspace by clicking Create an Empty Workspace.
- Copy an existing workspace to another name by clicking Create a Copy of this Workspace, and then selecting the workspace to copy from the selection box.
- Create a new workspace by importing a previously exported workspace by clicking Import a Workspace, and then find the workspace file to import by clicking the Browse button.

You can also import selected items from a workspace by doing the following:

1. Select Import Items from the File menu.
2. Select the workspace file (.wxp file) that contains the exported items. The file contents are displayed in the File Contents area.
3. Click the Import button to import the items. If there are items that already exist in your workspace, you will be prompted to rename them so that you do not overwrite existing jobs, data stores or custom functions.



Important: When importing a workspace, any Flow Engine machine definitions that were part of that workspace are also imported. These Flow Engine machines are added to the list of available Flow engine machines in the Architect tool.

You should review this list after importing a workspace to ensure that the new definitions are valid for your environment, and remove those definitions that are not valid.

On UNIX systems, invalid Flow Engine machines can cause errors to be written to the Formation Log Utility when a job is run. On Windows NT systems, invalid Flow Engine machines can cause the operating system to query the user for unknown computers when a job is run.

For example, if you import a workspace containing a Flow Engine machine for a computer that is not on your network, when you run that job on Windows NT you may receive an operating system warning that that computer is not available.

Opening a Workspace

The Formation product provides the Samples workspace as the default workspace. When you have created multiple workspaces for your Formation installation, you must select which workspace to use. You select the workspace to use either from the Formation Architect tool or from the Formation Workspace Manager tool.

To open a workspace from the Formation Architect tool, start the Formation Architect tool. The Open Workspace dialog is displayed before you can complete starting the Architect tool. Select which workspace to use from the Workspace box in the Open Workspace dialog and click the Open button.

To open a workspace from the Formation Workspace Manager tool, start the Formation Workspace Manager and click the Open button. The Formation Architect tool is started with the selected workspace open.

The Formation Architect tool and the Formation Workspace Manager tool cannot both be executing at the same time on the same computer.

Exporting a Workspace for Backup

You save a workspace to a file by exporting the workspace. You typically export a workspace when you want to:

- Save a backup copy of a workspace. It is a good practice to regularly backup your workspaces for safekeeping, just as you backup any information that would be difficult to recreate.
- Copy a workspace between Formation Architect installations. You export the workspace to copy from the first Formation Architect installation, copy the file to the new location, and then import the workspace file into the second Formation Architect installation.

You export a workspace by using the Formation Workspace Manager tool. You start the Workspace Manager by selecting Workspace Manager from the Start menu. You cannot start the Workspace Manager tool when the Architect tool is already executing. Exit the Architect tool before starting the Workspace Manager tool.

The Workspace Manager appears as in the following illustration.

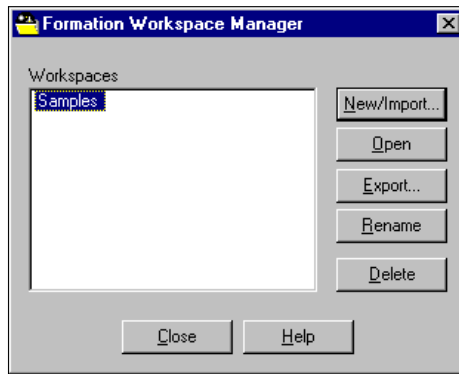


Figure 3-3
The Workspace Manager

You export a workspace as follows:

1. Select the workspace to export and click the Export button on the Formation Workspace Manager dialog. The Export Workspace dialog is displayed as in the following illustration.

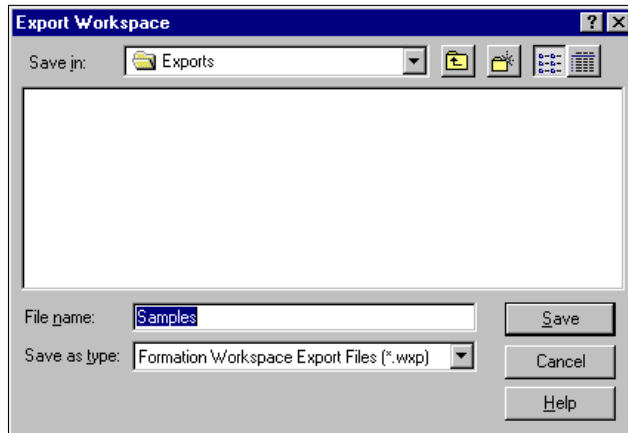


Figure 3-4
Exporting a Workspace

2. Select the directory location in which to place the exported workspace by navigating to that directory using the Save in box. By default, all workspaces are exported to the /Exports subdirectory of the directory in which the Formation Architect tool is installed.

3. Select or specify the file name of the workspace to export. By default it is the same as the name of the workspace. If you specify a name, make sure that it is a valid Windows NT filename.
4. Click the Save button to have the workspace exported to the directory and file you specified.

You can also export selected items from a workspace by doing the following:

1. Select Export Items from the File menu. The jobs, data stores, and custom functions in the current workspace are displayed in the Export dialog.
2. Select the items in the workspace you want to export and click the Export button.
3. Specify the name and location of the workspace file in which you want to save the workspace items.

For information on how to import a workspace that has already been exported, refer to [“Creating, Copying, or Importing a Workspace” on page 3-4](#).

Renaming and Deleting Workspaces

You rename and delete workspaces by using the Workspace Manager tool. To rename a workspace or delete a workspace, select Formation Workspace Manager from the Start menu.

To rename a workspace, do the following:

1. Select the workspace to be renamed from the list of workspaces.
2. Click the Rename button.
3. Provide a new workspace name in the selected area.

To delete a workspace, do the following:

1. Select the workspace to be deleted from the list of workspaces.
2. Click the Delete button. The workspace is removed from this Formation Architect installation.

You cannot delete a workspace if it is the only workspace available.

Using Jobs

Before you can create a data flow diagram that describes how to process and integrate your data, you must create a job to contain it. A job contains the data flow diagram that defines the data processing to be performed and also a set of runtime values that describe the environment in which the data processing will occur.

You manage the jobs in a workspace using the Manage Jobs dialog. When you start the Formation Architect tool, you select the job to use from the Manage Jobs dialog. The Manage Jobs dialog is displayed in the following illustration.

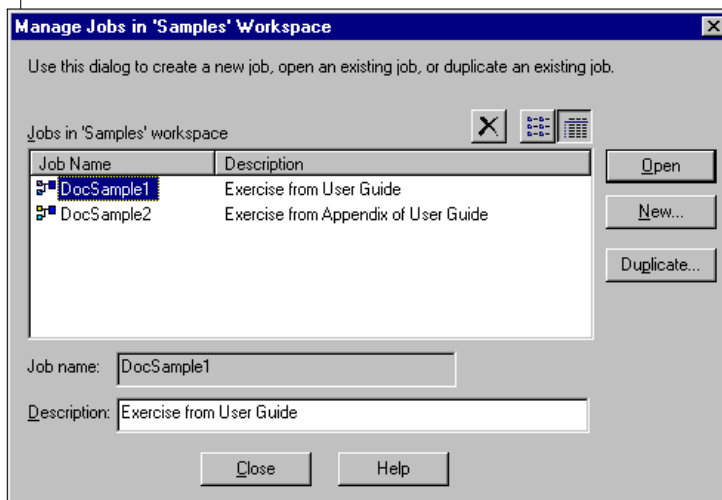


Figure 3-5
*The Manage Jobs
Dialog*

You can also access the Manage Jobs dialog while the Architect tool is executing by selecting Manage Jobs from the File menu of the Architect tool.

The following table lists the tasks you can perform to manage Formation jobs.

To...	...Do the Following in the Manage Jobs Dialog
Open an existing job	Select the job to open from the list of jobs and click the Open button.
Create a new job	Click the New button, and supply a name and description for the job in the New Job dialog.
Duplicate an existing job	Select the job to duplicate, click the Duplicate button, and supply a name and description for the duplicated job in the Duplicate Job dialog.
Delete a job	Select the job to delete and click the delete symbol (it resembles an X) button that is above the list of jobs.

To back up a job, you must export the workspace that contains the job. For more information on exporting a workspace, refer to [“Exporting a Workspace for Backup”](#) on page 3-7.

Defining the Data

Defining the Data	4-3
Overview of Creating Data Definitions	4-3
The Process of Creating Data Definitions	4-4
Understanding Data Stores.	4-5
Creating or Modifying Data Definitions Using the Data Definition Wizard	4-8
Data Definition Naming Rules	4-9
Considerations When Importing Schema	4-10
Modifying Data Definitions Within a Data Flow Diagram	4-10
Using Multibyte Data with Informix Formation	4-11
Using and Setting Locales	4-11
Using the Multibyte Text Data Type.	4-14
Copying Data Between Operating Systems	4-17

Defining the Data

This chapter describes how to define the data that will be used by a job.

This chapter is organized into the following sections:

- [Overview of Creating Data Definitions](#)
- [Creating or Modifying Data Definitions Using the Data Definition Wizard](#)
- [Modifying Data Definitions Within a Data Flow Diagram](#)
- [Using Multibyte Data with Informix Formation](#)
- [Copying Data Between Operating Systems](#)

Overview of Creating Data Definitions

When creating a job, you need to create data definitions for the data that will be input and output by the job. Formation supports both input and output from the following sources.

Input or Output Source	Operators
File	File Import File Export
Informix Dynamic Server	Informix DS Import Informix DS Export
Informix Dynamic Server/AD	Informix DS AD Import Informix DS AD Export

(1 of 2)

Input or Output Source	Operators
Microsoft SQL Server	MSSQL Server Import MSSQL Server Export
ODBC-compliant databases	ODBC Import ODBC Export
Oracle Server (7 or 8)	Oracle Server Import Oracle Server Export
Red Brick Warehouse	Red Brick Import Red Brick Export

(2 of 2)

In most cases, the database should be installed on the same machine as the Flow Engine component of the Formation product to enhance the performance of the job when it is reading to and writing from the database.

The Process of Creating Data Definitions

The following is the process you use to create data definitions for use in a job:

1. Identify the characteristics of the tables or records that will be input or output by the job, such as the relational database or file containing them, the fields used, and so on. As part of this step you should verify that all the field data types in the data are supported by the Formation product.

For a list of the supported data types, refer to [Appendix B, “Summary of Supported Data Types.”](#)

2. Create data stores to contain the data definitions for the tables or records you will use. A data store can contain either one or more table definitions from a relational database, or a record definition for a file.

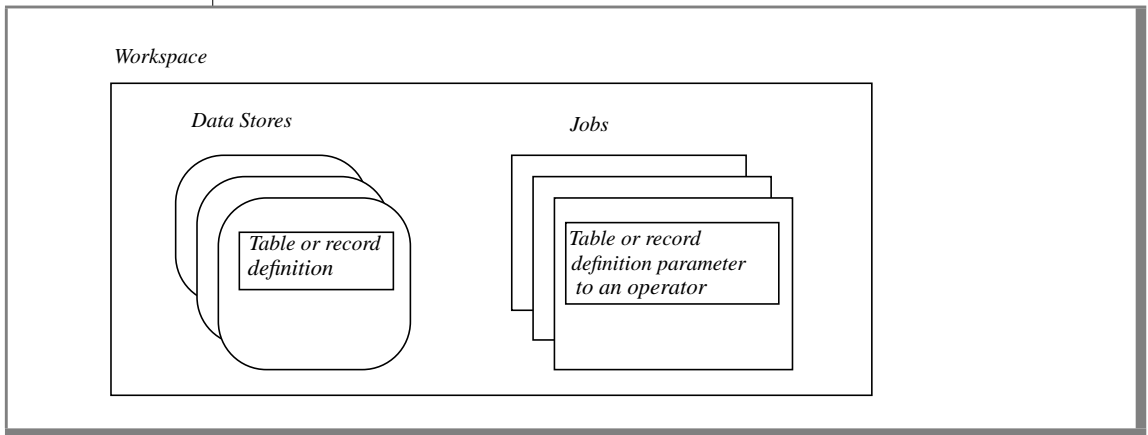
For more information on selecting or creating a data store, table definitions, or record definitions, refer to [“Understanding Data Stores.”](#)

Once the data store has been created, you specify the data definitions for the tables or records in the data store as parameters to the appropriate import or export operators in the data flow diagram for the job.

For more information on the supported import and export operators, refer to [Chapter 6, “Selecting and Using Operators.”](#) For more information on creating data flow diagrams, refer to [Chapter 5, “Creating Data Flow Diagrams.”](#)

Understanding Data Stores

To use a record or table definition in an import or export operator, you need to first define a data store to contain the record or table definition. Data stores are contained within workspaces just as are jobs. Table or record definitions are contained within data stores. The table or record definitions are then specified as parameters to import or export operators within jobs as shown in the following illustration.



There are two kinds of data stores:

- File data stores contain a single record that contains one or more fields (also known as columns). File data stores can contain fixed-length records, variable length delimited records, and ASCII, EBCDIC, or binary data.
- Relational database data stores contain one or more tables that contain one or more fields (columns)

Different types of data stores contain different types of data. For example, a file data store contains record definitions and an Informix Dynamic Server data store contains Informix Dynamic Server table definitions. You can place new table or record definitions in an existing data store of the same type or in a new data store of that type.

Data stores exist in a workspace so that they can be reused by multiple jobs.

To view the existing data stores in a workspace, select the Data Stores tab from the Navigator in the Formation Architect tool. The existing data stores are listed. To see the table or record definitions within a data store, click on the appropriate data store. To see the fields defined in a table or record definition, click on the appropriate table or record definition.



Tip: *It is a good idea to group jobs that use the same data stores into a single workspace so that you can reuse existing data stores.*

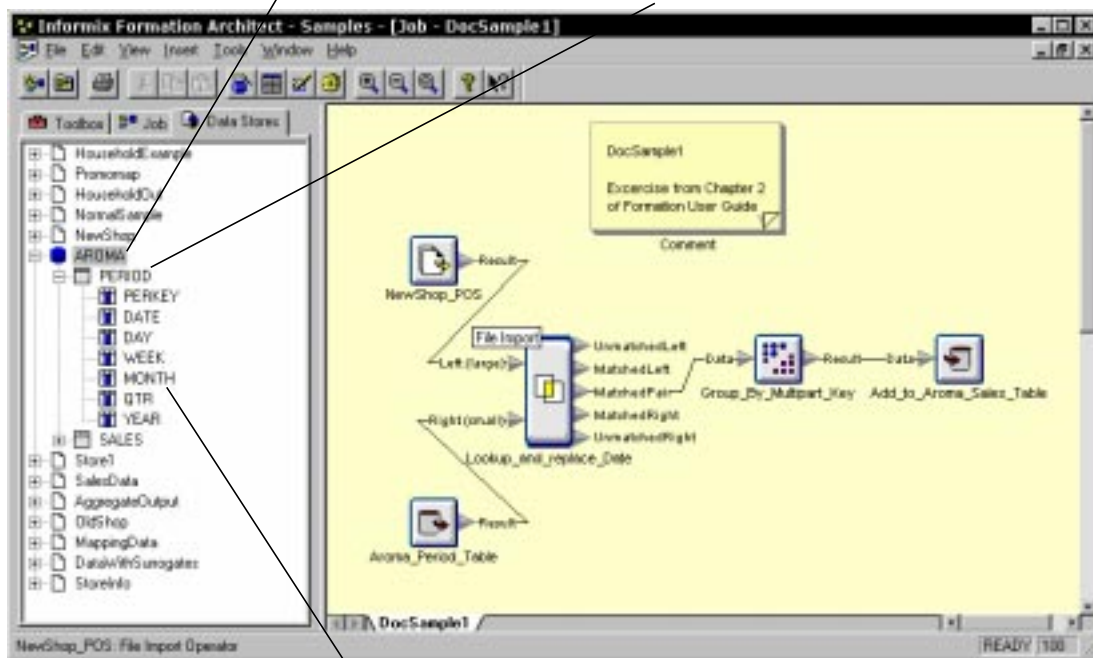
To edit the data store, table definitions, or record definitions double click on the item you want to edit in the Data Stores tab.

The following illustration shows the data stores used by the job DocSample1 in the Samples workspace.

Figure 4-1
Job DocSample1

AROMA Data Store Used by the DocSample1 Job

Table PERIOD in AROMA Data Store



Field MONTH in Table PERIOD

Creating or Modifying Data Definitions Using the Data Definition Wizard

To create or modify a data store, table definition, or record definition, you use the Data Definition Wizard. The Data Definition Wizard allows you to manually create data definitions or to automatically create data definitions by importing schemas from existing relational databases. Whether you are creating the definitions manually or automatically, be sure that the definitions use the data types supported for the Formation product. For a list of the supported data types, refer to [Appendix B, “Summary of Supported Data Types.”](#)

To use the Data Definition Wizard, select **Data Definition Wizard** from the Tools menu. The Data Definition Wizard is displayed. Follow the instructions in the wizard to create or modify a data store, table or record definition. The following is an illustration of the initial Data Definition Wizard dialog.

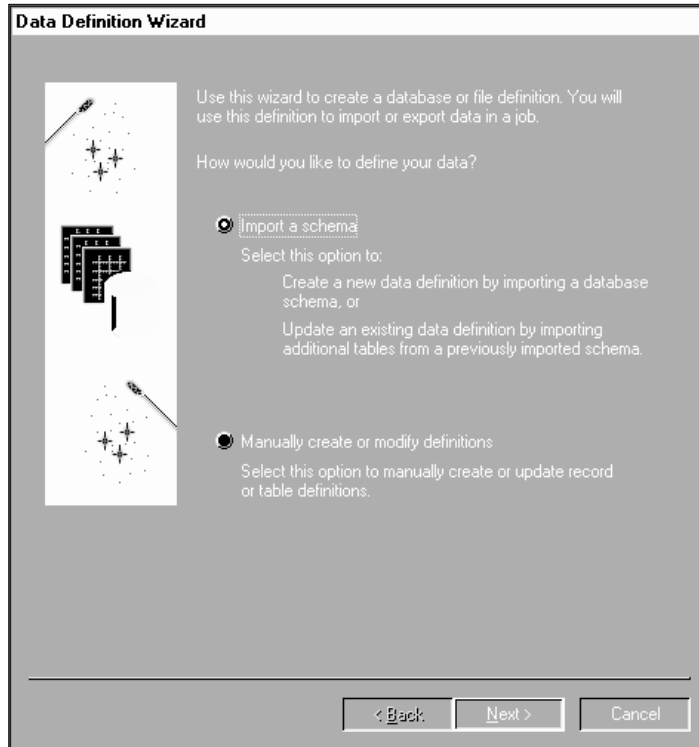


Figure 4-2
*The Initial Data
Definition Wizard
Dialog*

Data Definition Naming Rules

When you enter data store, table definition, record definition, or field definition names, those names must meet the following rules:

- If this is a new name, it must be unique.
- The name cannot contain spaces, slashes, backslashes, or vertical bars (pipes).
- Do not use C++ keywords as names. For a list of these keywords, refer to [“C++ Reserved Words Were Used as Identifiers in the Job.”](#)

- If you are manually defining a database field name, that name must match the relational database column name exactly, including the case of each letter.

Considerations When Importing Schema

The following are considerations you should take into account when importing schema using the Data Definition Wizard:

- When you use the Data Definition Wizard to define a data store by importing the schema of a relational database, you make use of an ODBC driver for that database. These ODBC drivers are described in the [Informix Formation Installation and Configuration Guide](#), or the appropriate ODBC driver documentation set.
- Table synonyms are not supported.

Modifying Data Definitions Within a Data Flow Diagram

Occasionally, you may need to edit fields in data definitions within a data flow diagram. To view or modify the fields in a data definition within a data flow diagram, you use the *Data Definition Editor* as follows:

1. Place your cursor over the port of the operator that contains the data definition you want to view or edit and click the right mouse button to display the context menu.
2. Select Define from the context menu, and the Data Definition Editor is displayed for that data definition.
3. View or edit the data definition.
 - If the Data Definition Editor title bar displays “Data Definition Editor”, you can edit the data definition.
 - If the Data Definition Editor title bar displays “Data Definition (Read Only),” you may only view the data definition and not edit it.

Using Multibyte Data with Informix Formation

Informix Formation supports the use of multibyte data in the English and Japanese languages as follows:

- Importing multibyte data
- Exporting multibyte data
- Manipulate multibyte text using the text manipulation functions
- Translate between multibyte encodings

The following are some of the general restrictions in using multibyte data in Informix Formation:

- All names and text parameters used in the Formation Architect must be in ASCII characters. Such names and parameters include the following: names of workspaces, jobs, data stores, tables, fields, operators, custom objects, file names used in the File Export and File Import operators, text entered in visual snippet comments or as parameters to operators, and so on.
- In the Formation Architect, all status messages are output in ASCII characters. In the Formation Flow Engine, all status messages are output in ASCII characters. If the status message includes a sample of data that is being manipulated, that data will remain in its original form, which may be in multibyte characters.
- Any operations performed on multibyte data will perform more poorly than performing those same operations on ASCII data.

The following sections describe using multibyte data with Informix Formation.

Using and Setting Locales

Multibyte character sets have locales associated with them. A locale is a keyword indicating the linguistic or regional rules to use when manipulating data. All the supported locales support ASCII. Unicode is not supported.

You set locales in Informix Formation as described in the following table.

Items with Locales	Description
Workspaces	<p>When a workspace is created, the workspace is assigned the locale of the Windows NT computer on which it was created. You set the Windows NT default locale for your computer using the Regional Settings applet in the Control Panel.</p> <p>When a workspace is opened, the Formation Architect checks the current default locale on that computer. If the default locale does not match the workspace locale, a warning is given and the user must change the locale settings on the computer to the locale of the workspace to open that workspace.</p> <p>The following are not supported:</p> <ul style="list-style-type: none">■ Conversion between workspaces with different locales■ Importing jobs into the current workspace whose locale does not match the locale of the current workspace
Data stores	<p>You set the locale using the Data Wizard either when you import schema or manually define a data store.</p> <p>The Locale field is displayed when the Type of the data store is set to any value other than Flat File or Text File with a Character Set value of Multi-Byte.</p> <p>Each data store supports one locale, and so all multibyte fields in that data store are also in one locale. You cannot mix tables with different locales in a single data store definition.</p>
Flow Engine Machines	<p>You set the locale using the Configure Flow Engine Maching dialog.</p> <p>For each Flow Engine machine, including the default local machine, the user must set a default Flow Engine machine locale before generating the job files.</p>

Locales in Informix Formation have the following format.

[Language]_[Territory].[CodePage]@[SortAlgorithm]

The following are the supported locales:

- English_UnitedStates.US-ASCII@Binary
- English_UnitedStates.US-ASCII@Default
- English_UnitedStates.Latin1@Binary

- English_UnitedStates.Latin1@Default
- English_UnitedStates.MS1252@Binary
- English_UnitedStates.MS1252@Default
- Japanese_Japan.JapanEUC@Binary
- Japanese_Japan.JapanEUC@Default
- Japanese_Japan.MS932@Binary
- Japanese_Japan.MS932@Default

The CodePage portion of the locale keyword is described in the following table.

CodePage Keyword	Description
US-ASCII	7-bit ASCII
Latin1	ISO8859-1 Western European
JapanEUC	Japanese Extended UNIX code, this includes JIS X 0212
MS932	Microsoft Windows Japanese, this is a superset of Shift-JIS which is the default codepage on Japanese Windows NT
MS1252	MS Windows Latin1 (ANSI), this is a superset of Latin1 which is the default codepage on English Windows NT

You have the ability to define a multibyte string constant in the Architect. All multibyte strings entered by users are in a workspace locale and are converted from the workspace locale to the default Flow Engine machine locale during code generation time. The users will see a validation error if conversion between the two locales fails. For example, if the current workspace locale is Shift-JIS and the default machine locale is JapanEUC, if the user types in some characters that are only found in Shift-JIS and cannot be converted to JapanEUC, an error will be reported at job validation time.

Using the Multibyte Text Data Type

The Multibyte Text data type is used to support the use of multibyte data. This data type is locale-sensitive and supports most of the text manipulation functions, such as search, compare, substitute, and so on. In the Formation Architect, the ASCII text data type is named Text, and the multibyte text data type is named Multibyte Text.

Comparison sorting functions using the Multibyte Text data type are all collation-based. Conversions between two multibyte text fields with different locales is supported. Conversions between a multibyte text and other data types is also supported as shown in the following table.

	INT	UINT	FLT	DBL	DEC	DATE	TIME	DATETIME	INTERVAL	TEXT
To	yes	yes	yes	yes	yes	Only ASCII	Only ASCII	Only ASCII	Only ASCII	yes
From	yes	yes	yes	yes	yes	Only ASCII	Only ASCII	Only ASCII	Only ASCII	yes

You can also compare two multibyte text strings with different locales directly. Since all multibyte string functions are locale based, you should be aware that operations on the multibyte data could be slow, particularly when conversions between locales is required.

The existing text data type is reserved to support ASCII data and allows processing of ASCII-only data to be optimized. If you know that your data is in ASCII, it is recommended that you use the ASCII text data type instead of the multibyte text type.

When using multibyte text strings, the null byte is always treated as the terminator of the string. Formation does not support any encoding with null bytes as part of a character.

For all the character types in the databases (excluding NCHAR, NVARCHAR in Informix Dynamic Server/AD, and NCHAR and NVARCHAR2 in Oracle8), the mappings from these types to Formation's internal data types are locale based. That is, if the database locale is ASCII, all these data types in the database are mapped to Formation's internal ASCII text type. If the database locale is not ASCII, all these character types are mapped to Formation's interval multibyte text type.

The following table lists the supported character data types and corresponding databases.

Database	Character Data Types
Informix Dynamic Server	CHAR/VARCHAR NCHAR/NVARCHAR(mapping is not locale sensitive)
Informix Dynamic Server/AD	CHAR/VARCHAR NCHAR/NVARCHAR(mapping is not locale sensitive)
Microsoft SQL Server	CHAR/VARCHAR
ODBC	SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR
Oracle7 Server	CHAR/VARCHAR2
Oracle8 Server	CHAR/VARCHAR2 NCHAR/NVARCHAR2(mapping is not locale sensitive)
Red Brick Warehouse	CHAR/CHARACTER

The NCHAR, NVARCHAR2 data types in Oracle 8, and the NCHAR and NVARCHAR data types in Informix Dynamic Server and Informix Dynamic Server/AD are always mapped to the multibyte text type. If you have an Informix Dynamic Server or Informix Dynamic Server/AD database with NCHAR and NVARCHAR data types that only stores ASCII data because that database is not a NLS version, text operations on that ASCII data could perform poorly since they are mapped to the multibyte text data type and corresponding multibyte text functions are used.

The following visual snippet text functions support both the ASCII and the multibyte data types, with the comments listed.

Text Function Name	Comments
Char	Only the ASCII character is supported. But the output could be in a multibyte locale code page if ASCII is included in that code page.
Text Compare	
Text Concatenate	
Text Downcase	
Text Format	Users could mix ASCII and multibyte data types on inputs, but the format flag has to be set to %s.
Text Length	
Text Search & Split	
Text Search In List	
Text Split	
Text Substitute	
Text Trim	
Text Uppcase	

You can mix ASCII and multibyte data types for the inputs of the text functions. For example, you can connect an ASCII field to one input port of the Text Compare function and a multibyte field to the other input port of the same snippet function. Conversions between the text fields are handled automatically in the Architect. However, the following rules apply to the text snippet functions when inputs have mixed ASCII and multibyte data types:

- If any input port is connected to a multibyte field, the output port is set to the multibyte text data type (if the port is a text output port).
- If all input ports are connected to an ASCII field, the output port is set to the ASCII text data type (if the port is a text output port).

- Conversions between numeric types and multibyte and ASCII text types are based on the output port type. If the output port is set to multibyte type, conversions are from numeric to multibyte. If the output port is set to ASCII type, conversions are from numeric to ASCII.
- Conversions between Multibyte Text and the Date, Time, Timestamp and Interval data types are restricted to ASCII characters. No multibyte character month names, year name, and so on are supported in conversions.

Copying Data Between Operating Systems

If you move the file data you want to import or export between operating systems, you may encounter data conversion problems that will cause problems in your job. In general, you should use supported tools for converting your files between incompatible operating systems, such as UNIX and Windows NT.

On Sun Solaris, use the `dos2unix` utility to convert Windows NT files to UNIX files, or use the `unix2dos` utility to convert UNIX files to Windows NT files.

Creating Data Flow Diagrams

In This Chapter	5-3
Overview of Data Flow Diagrams.	5-3
Defining the Input and Output Data Stores	5-6
Selecting the Import and Export Operators	5-6
Specifying the Data Store	5-7
Specifying the Data Location	5-7
Selecting the Transformation Operators.	5-7
Connecting the Operators	5-8
Data Propagation with Import or Export Operators	5-9
Data Propagation with Transformation Operators	5-9
Providing Operator Parameters	5-10
Validating the Job	5-11

In This Chapter

This chapter describes how to create a data flow diagram. The information in this chapter is organized into the following sections:

- [Overview of Data Flow Diagrams](#)
- [Defining the Input and Output Data Stores](#)
- [Selecting the Import and Export Operators](#)
- [Selecting the Transformation Operators](#)
- [Connecting the Operators](#)
- [Providing Operator Parameters](#)
- [Validating the Job](#)

Overview of Data Flow Diagrams

A data flow diagram expresses the integration and processing of the data using a visual editing approach. You create a data flow diagram by connecting together visual representations of computing tasks, called *operators*. You create data flow diagrams by selecting operators that perform the processing you need and connecting the data flows between them in the correct processing order. For example, a data flow diagram is created when you connect a data flow from the output port of a File Import operator to the input port of the Filter operator. One or more data flow diagrams can be defined within the context of a job.

The following is an illustration of the data flow diagram in the DocSample1 job in the Samples workspace.

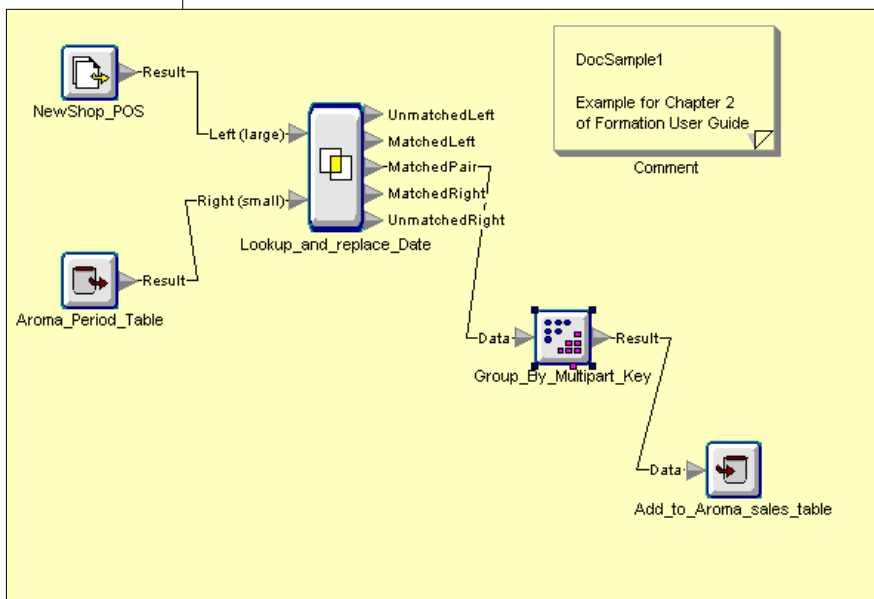


Figure 5-1
DocSample1 Data Flow Diagram

The following list outlines the methodology for creating a data flow diagram. Each step in the methodology is explained in detail in this chapter or in another chapter:

1. Create a job in a workspace to contain your new data flow diagram. You should create this job in a workspace other than the Samples workspace. Making changes to the Samples workspace may make it no longer match the descriptions of it in the online help and print documentation, and make it less useful to you.

For more information on creating jobs or workspaces, refer to [Chapter 3, “Using Workspaces and Jobs.”](#)

2. Create the data stores that describe the source and target data for the job.

For more information on creating data stores, [Chapter 5, “Creating Data Flow Diagrams.”](#)

3. Select the import and export operators that represent the sources coming into the job and the targets created by the job and associate data stores with each of these operators.

For more information on selecting import and export operators, [“Selecting the Import and Export Operators” on page 5-6.](#)

4. Add the transformation operators that specify the work to be performed on the data and the order in which that work should be performed.

For more information on selecting transformation operators, [“Selecting the Transformation Operators” on page 5-7.](#)

5. Connect the operators together to create the data flow diagram.

For more information on connecting operators, [“Connecting the Operators” on page 5-8.](#)

6. Add parameter values to each transformation operator. These parameters describe the logic and limits for the operation represented by the operator icon.

For more information on operator parameters, [“Providing Operator Parameters” on page 5-10.](#)

7. Ensure that the data flow diagram passes validation.

For more information on validation, [“Validating the Job” on page 5-11.](#)

Defining the Input and Output Data Stores

Data stores define the structure of data and can represent either database tables or text or binary files. Before you can use data stores in a job, you must either create their definitions or, in the case of database tables, import them from a database. Since data stores in a workspace can be used by any job in that workspace, you do not have to create new data stores for each job if you use the same tables or files as inputs or outputs in a series of jobs.

The following list describes the steps you perform to define the data stores that will be used in the data flow diagram:

1. Identify the tables or records that will be input or output by the data flow diagram.
As part of this step, verify that all the fields in the data are supported by the Formation product. For a list of the supported data types, refer to [Appendix B, “Summary of Supported Data Types.”](#)
2. Import or create one or more table or record definitions. These definitions are contained in *data stores*.

For more information on selecting or creating a data store, table definitions, or record definitions, refer to [Chapter 4, “Defining the Data.”](#)

Selecting the Import and Export Operators

After you have defined the data stores, select the import and export operators that represent the sources and targets in the job. *Import operators* specify the source files or tables where data enters the job. *Export operators* define the target locations for data storage when the job is finished. You can use any number of import and export operators in a data flow diagram, including multiple sources and multiple targets.

To add an operator to the data flow diagram, drag it from the Toolbox tab of the Navigator to the job window.

After you add the import and export operators to the data flow diagram, you must specify values for the parameters that select the data store and specify the location of the data.

Specifying the Data Store

The data store you select depends on whether the operator will handle data from a file or from a database table:

- When the operator represents a file, the parameters dialog displays a Record Definition parameter.
- When the operator represents a table, the parameters dialog displays a Table Definition parameter.

To select the data store, do the following:

1. Click the Define button to the right of the Record Definition or Table Definition parameter. A dialog is displayed, listing the data stores that match the type of the operator. For example, for an Oracle Import operator, the dialog lists only Oracle data stores.
2. Select the record or table definition that defines the structure of the data used by that operator and click OK.

Specifying the Data Location

You must specify where the source data is located or where you want to store the target data.

- When the operator represents a file, type the path and file name of the data file. For detailed information on the path formats available, refer to [“File List Parameter Formats” on page 6-12](#).
- When the operator represents a table, specify the table and database information as required by the parameters dialog for that operator.

Selecting the Transformation Operators

After you have defined the sources and targets for the job by setting up the import and export operators, you define how the transformations that should occur. The job can contain one or many transformations that cleanse, modify, delete, edit or otherwise manipulate the data. Each transformation, such as a Group By, Join, or Sort, is represented by a single operator.

You define how to transform the data by selecting operators to perform the computing tasks you need to perform, and then placing them in the correct order in the data flow diagram. This forms the skeleton of the data flow diagram. Refer to the data flow diagrams provided as part of the Samples workspace for examples of how operators are used together.

For information on selecting and using operators, refer to [Chapter 6, “Selecting and Using Operators,”](#) or consult the Formation Architect online help.

Connecting the Operators

After you have selected the import and export operators and the transformation operators to use, you draw connections from one operator to the next in the data flow.

To draw a connection, click on the operator port where you want to begin the line, and then drag the line to the operator port where you want the data to go. You can connect the data flow coming out of an operator to more than one operator by repeating this process. This will send a copy of the data down each data flow, so that the same data at the starting point is processed in different ways.

To delete a connection between two operators, click on that part of the data flow and either press the Delete key or select Delete from the Edit menu.



Important: For best results, you should define the parameters for an import or export operator before you connect it to other operators in the data flow diagram.

When you connect the operators in a data flow diagram, data definitions are copied from one operator to the next, so that you do not have to manually select data definitions at each port on each operator in the data flow diagram. For example, when you connect a File Import operator to a Sort operator, the data definition you selected for the File Import (using the Record Definition parameter) is automatically sent to the Sort operator.

In most cases, providing data definitions for import and export operators is all you have to do, since this data propagation process automatically copies the data definitions throughout the data flow diagram.

Data Propagation with Import or Export Operators

When a data store is changed, the new data definition is automatically propagated through the data flow diagram, possibly causing conflicts with operators that are using the previous definition. Use the Propagate Data Definition command on the context menu for the import or export operator ports to propagate the new data definition towards the red line.

Data Propagation with Transformation Operators

You can also encounter data propagation problems without changing the data store by manipulating operators in a data flow diagram.

How you handle the problem depends on whether the operator has the same data definition at both the input and output ports or if the operator has different data definitions at the input and output ports:

- If the problem operator has the same data definition at both the input and output ports the operator is simply processing records without changing the structure of the data. Sort and Deduplicate are examples of this kind of operator.

To correct the problem, check the data definitions for the ports on both sides of the red line and decide which is the correct data definition to use. Select that port and execute Propagate Data Definition on the context menu.

If there are multiple red lines, first correct the red line closest to an import or export operator and then work towards the rest of the red lines. As you correct the data definitions, the corrected data definitions will be propagated, and so may reduce the amount of changes you need to manually make.

- If the problem operator has different data definitions at the input and output ports, the operator is changing the structure of the data.

Operators that have different data definitions on the input and output ports tend to be more likely to cause and have data propagation problems than pass-through operators. Advanced Join, Aggregate, Cross Product Join, Cursor, First Normal, Group By, Household, Join, Program, and Project are examples of this kind of operator.

Typically, it is the output data flow of the operator that is incorrect and displayed in red. You can correct this problem by deleting that data flow and redrawing the connection to re-propagate the data definition into the operator. If that does not correct the problem, edit the problem data definition on the port by selecting Define on the context menu for that operator port.

Providing Operator Parameters

After you have connected the operators in the data flow diagram, you specify the parameters for each transformation operator.

Many of the operators require you to provide some programmatic logic that describes how you want data in that operator to be processed. This programmatic logic is called a *snippet* in the Formation product.

There are two kinds of snippets:

- Visual snippets
- Code snippets

For more information on creating visual or code snippets, refer to [Chapter 7, “Working with Snippets.”](#)

Validating the Job

The Architect saves your work automatically as you construct your data flow diagram. Once you are done creating the data flow diagram, you are ready to verify that the data flow diagram is syntactically correct by validating it.

To validate a data flow diagram, select **Validate Job** from the **Tools** menu.

If the job contains no errors, such as missing value for required parameters, you will see a message that validation was successful.

If the job contains errors, a dialog displays the errors found and the locations for those errors. You can double-click on an error to highlight its location in the job, or in the case of a visual snippet, you can open the snippet for editing.

Selecting and Using Operators

In This Chapter	6-5
Overview of Operators	6-6
Operator Ports	6-6
Operator Parameters	6-7
The Parameters Dialog	6-7
Visual Snippets	6-8
Code Snippets	6-8
Import and Export Operators	6-8
Importing Data	6-8
Exporting Data	6-9
Synchronizing Imports and Exports Using Execution Order	6-10
Quick List of Import and Export Operators	6-10
File Import Operator	6-11
File Export Operator	6-12
Notes on Using File Import and File Export	6-12
File List Parameter Formats	6-12
Using Pipes in the File List Parameter.	6-13
Informix DS Import Operator	6-13
Informix DS Export Operator	6-13
Informix DS AD Import Operator	6-14
Informix DS AD Export Operator	6-15
MSSQL Server Import Operator	6-16
MSSQL Server Export Operator	6-16
ODBC Import Operator	6-17
ODBC Export Operator	6-17
Oracle Server Import Operator	6-18
Oracle Server Export Operator	6-18

Red Brick Import Operator	6-19
Red Brick Export Operator	6-19
Simple Transformation Operators	6-20
Deduplicate Operator.	6-20
Using Deduplicate	6-21
Sample Job	6-21
Filter Operator	6-21
Sample Job	6-21
Project Operator	6-22
Sample Job	6-22
Sort Operator	6-22
Sample Job	6-23
Union Operator.	6-23
Sample Job	6-23
Join Transformation Operators	6-24
Input Ports	6-24
Output Ports	6-24
Join Operator	6-25
Using Join	6-25
Sample Job	6-25
Advanced Join Operator.	6-25
Using Advanced Join	6-26
Cross Product Join Operator	6-27
Using Cross Product Join	6-27
Example	6-27
Advanced Transformation Operators.	6-29
Aggregate Operator	6-29
Using Aggregate	6-30
Sample Job	6-30
Cursor Operator	6-30
Using Cursor	6-31
First Normal Operator	6-31
Using First Normal	6-32
Sample Job	6-32
Group By Operator	6-32
Using Group By	6-32
Sample Job	6-33
Household Operator	6-33

Using Household	6-34
Sample Job	6-34
Program Operator	6-34
Split Operator	6-36
Using Split.	6-37
Sample Job	6-37
Surrogate Key Operator	6-37
Sample Job	6-38
Data Parallelism Operators.	6-38
Partition Operator	6-38
Using Partition	6-39
Gather Operator	6-39
Using Gather	6-39
Performance Considerations	6-40

In This Chapter

This chapter provides an overview of how to use operators in data flow diagrams. This chapter includes the following sections:

- [Overview of Operators](#)
- [Import and Export Operators](#)
- [Simple Transformation Operators](#)
- [Join Transformation Operators](#)
- [Advanced Transformation Operators](#)
- [Data Parallelism Operators](#)
- [Performance Considerations](#)

The Informix Formation Architect online help system contains complete descriptions of the operator ports and parameters. To see context-sensitive help for an operator while in a data flow diagram, select the operator and press the F1 key.

The Samples workspace provided with the Formation product contains sample jobs for many operators discussed in this chapter.

Overview of Operators

Data flow diagrams are composed of operators connected by data flows. Each operator indicates a unit of data processing, such as joining or aggregation. Operators are grouped by function into the following folders in the Navigator:

Operator Folder	Description
Import Operators	Contains operators used to move data from an external source, such as a database table or a file, to a data flow diagram.
Export Operators	Contains operators used to move data from a data flow diagram to a file or a database table.
Transformation Operators	Contains operators used to manipulate and transform data within a data flow diagram.

To see these folders, click on the Operators folder in the Toolbox tab of the Navigator. To see the operators within each folder, click on the plus sign next to the folder.

The name of an operator in a data flow diagram is either the default name (the name of the operator followed by an underscore and a number) or a user-defined name. For example, Join_1 is a system-defined name for a Join operator, and Shop2_POS is a user-defined name.

Operator Ports

A port directs the flow of data into or out of an operator. An operator can have one or more input ports and output ports.

- Input ports bring data on the data flow into the operator for processing.
- Output ports send data from the operator back into the data flow.

For example, an import operator, which brings data into the job, has no input ports and one output port, because the data flow starts with an import operator.

In most cases, the input port is named Data and the output port is named Result. Some operators use different port names to indicate special-purpose ports, such as the Valid and Invalid output ports on the Filter operator and the UnmatchedLeft port on the join operators.

Operator Parameters

Each operator has one or more parameters that control how the operator works on the data. For example, an import operator has a parameter that specifies where the data file is located.

The Parameters Dialog

To see the parameters associated with an operator, double-click on the operator in a data flow diagram. The following graphic shows the Parameters dialog for a Deduplicate operator named UniquePromoCodes.

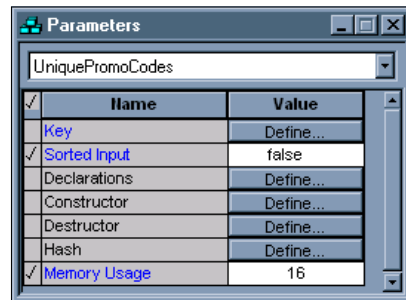


Figure 6-1
The Parameters Dialog

The Parameters dialog displays the following information about the parameters for with an operator:

- The leftmost column displays a checkmark (✓) if the parameter has been assigned a value or is blank if no value has been assigned. The Formation software supplies default values for some parameters, so you can see checkmarks when you open the parameter dialog for a new operator.
- The Name column lists the name of the parameter. Parameter names shown in blue are required. Parameter names shown in black are optional.

- The Value column displays the current value of the parameter. The Define button indicates a parameter that must be defined using a secondary dialog.

Visual Snippets

Some parameters use *visual snippets* to display programming logic. The visual snippet window opens automatically when you select a parameter that requires a visual snippet.

Refer to [Chapter 7, “Working with Snippets,”](#) for more information.

Code Snippets

When a parameter requires actual programming code, such as a call to a third-party program, the parameter is called a *code snippet*. Code snippets are displayed when you double-click on a parameter that requires a code snippet or when you choose Code Snippet Editor on the View menu.

Refer to [“Code Snippets” on page 7-21](#) for more information.

Import and Export Operators

To bring data into a job and store it at the end of a job, you specify import and export operators, respectively.

Importing Data

An import operator describes the source data that the job will use. If the data source is a file, you must specify a record definition that describes the structure of the data and specifies the location where the data file is stored. If the data source is a table in a database, you must specify a table definition that describes the data and provide the database and table information required to access the table.

Supported import formats include the following:

- ASCII files
- EBCDIC files
- Binary files
- Red Brick Warehouse
- Oracle7 Server
- Oracle8 Server
- Any database that supports ODBC
- Informix Dynamic Server
- Informix Dynamic Server/AD
- Microsoft SQL Server

Exporting Data

An export operator describes the target data information, including a record or table definition and the location of the file or database where the data should be stored upon job completion.

Supported export formats include the following:

- ASCII files
- EBCDIC files
- Binary files
- Red Brick Warehouse
- Oracle7 Server
- Oracle8 Server
- Any database that supports ODBC
- Informix Dynamic Server
- Informix Dynamic Server/AD
- Microsoft SQL Server

Synchronizing Imports and Exports Using Execution Order

You can synchronize import and export operations by using the Execution Order parameter that is required for all import and export operators.

If the Execution Order parameter is set to the default value of 0, the operator is not synchronized with other import and export operators.

If the Execution Order parameter is set to a value from 1 to 100, this indicates that the operator should be synchronized with other import and export operators. Operators that are synchronized are executed from lowest Execution Order value to highest.

For example, an import or export operator with an Execution Order value of 1 would complete before an import or export operator with an Execution Order value of 2 would begin. If a job is loading dimension and fact tables, typically you should set the Execution Order to 1 for dimension tables and to 2 for fact tables so that the dimension tables are fully loaded before the fact tables are loaded to ensure referential integrity.

Quick List of Import and Export Operators

The following table lists the import and export operators available in Formation.

Operator	Moves Data From	Moves Data To
File Import	File	Data flow
File Export	Data flow	File
Informix DS Import	Table in an Informix Dynamic Server database	Data flow
Informix DS Export	Data flow	Table in an Informix Dynamic Server database
Informix DS AD Import	Table in an Informix Dynamic Server/AD database	Data flow

(1 of 2)

Operator	Moves Data From	Moves Data To
Informix DS AD Export	Data flow	Table in an Informix Dynamic Server/AD database
MSSQL Server Import	Table in a Microsoft SQL Server database	Data flow
MSSQL Server Export	Data flow	Table in a Microsoft SQL Server database
ODBC Import	Table in a database supported by an ODBC driver	Data flow
ODBC Export	Data flow	Table in a database supported by an ODBC driver
Oracle Server Import	Table in an Oracle Server database	Data flow
Oracle Server Export	Data flow	Table in an Oracle Server database
Red Brick Import	Table in a Red Brick Warehouse database	Data flow
Red Brick Export	Data flow	Table in a Red Brick Warehouse database
Program	Moves data either from or to a data flow, or both, depending on the executable used	

(2 of 2)

File Import Operator

The File Import operator reads the data from one or more files and converts the data into a data flow. This operator can process ASCII files, EBCDIC files, or flat files (binary and ASCII in combination).

To use this operator in a job, you must specify values for the Record Definition parameter (either a record or table defined as a data store) and the File List parameter, which specifies the location of the source data. By default, the operator appends the data to any existing data at the target location.

File Export Operator

The File Export operator takes the data flow from the input port, performs any necessary data type conversions, and places the data into the file(s) specified in the File List parameter. This operator can process ASCII files, EBCDIC files, or flat files (binary and ASCII in combination).

To use this operator in a job, you must specify values for the Record Definition parameter (either a record or table defined as a data store) and the File List parameter, which specifies the location of the target for the transformed data. By default, the operator will append the data to any existing data at the target location.

Notes on Using File Import and File Export

File List Parameter Formats

The format for the File List parameter depends on whether the file you are specifying is stored on a Windows NT system or on a UNIX system.

The File List parameter supports the following formats when specifying files stored on the Windows NT platform:

```
file:///path\filename.ext
pipe:///path\filename.ext
file:///..\relative\filename.extrelative path name
\temp\filename.ext
.\filename.ext
..\temp\filename.ext
subdir\filename.ext
```

The File List parameter supports the following formats when specifying files stored on UNIX platforms:

```
file:///path/path/filename.ext
pipe:///path/path/filename.ext
/temp/filename.ext
./filename.ext
../temp/filename.ext
subdir/filename.ext
```

Using Pipes in the File List Parameter

You can use pipes to stream the output of one job to become the input of another job. You must specify exactly the same pipe name and path in the File List parameter of the File Export operator in the first job as you specify in the File List parameter of the File Import operator of the second job.

You must specify the full pathname to the pipe rather than using a relative path name.

Note that you should not run the second job before the first job has created the pipe; otherwise, the second job will fail with the message that it could not open the specified pipe.

Informix DS Import Operator

The Informix DS Import operator writes a table from an Informix Dynamic Server database to a data flow.

To use this operator, you must specify the following:

- A table in the data store that describes the structure of a table in the Informix Dynamic Server database.
- The database server where the Informix Dynamic Server database is located.
- The Informix Dynamic Server database to access.
- A valid user name for the database.
- The matching password for the user name, if it is needed.
- The number of parallel data flows to be used by this operator. The default and only possible value is 1.
- The execution order of this operator.
- The field delimiter character to use. The default is the vertical bar (|).

Informix DS Export Operator

The Informix DS Export operator writes records from a data flow to a table in an Informix Dynamic Server database.

To use this operator, you must specify the following:

- A table in the data store that describes the structure of a table in the database.
- The database server where the Informix Dynamic Server database is located.
- The Informix Dynamic Server database to access.
- A valid user name for the database.
- The matching password for the user name, if it is needed.
- The number of parallel data flows to be used by this operator. The default and only possible value is 1.
- The execution order of this operator.
- The field delimiter character to use. The default is the vertical bar (|).

When using the Informix DS Export operator, avoid using the optional Where Clause parameter to delete large tables since this can result in poor performance of your Formation job.

Informix DS AD Import Operator

The Informix DS AD Import operator writes a table from an Informix Dynamic Server/AD database to a data flow.

To use this operator, you must specify the following:

- A table in the data store that describes the structure of a table in the database.
- The ODBC data source to be used. The ODBC connection to the database is used to send the SQL statements that start the Informix Dynamic Server/AD loader. Once the loader is started, all data is transmitted directly between Formation and the loader; no data is transmitted using the ODBC connection.
- The name of the table in the database to be accessed.
- A valid user name for the database.
- The matching password for the User Name parameter.

- How the database handles ASCII special characters embedded in ASCII-text-based data files. The default value, FALSE, indicates that the database does not create embedded hexadecimal characters in text fields. If the value is TRUE, the database does create embedded hexadecimal characters in text fields.
- The number of parallel data flows to be created by this operator. The range is 1 to 32; the default value is 1.
- The field delimiter character to use. The default character is the vertical bar (|).

Informix DS AD Export Operator

The Informix DS AD Export operator writes records from a data flow to a table in an Informix Dynamic Server/AD database.

To use this operator, you must specify the following:

- A table in the data store that describes the structure of a table in the database.
- The ODBC data source to be used. The ODBC connection to the database is used to send the SQL statements that start the Informix Dynamic Server/AD loader. Once the loader is started, all data is transmitted directly between Formation and the loader; no data is transmitted using the ODBC connection.
- The name of the table in the database to be accessed.
- A valid user name for the database.
- The matching password for the User Name parameter.
- How the database handles ASCII special characters embedded in ASCII-text-based data files. The default value, FALSE, indicates that character fields in text data files are not checked for embedded special characters. If the value is TRUE, the fields are checked for embedded special characters.
- The number of parallel data flows to be created by this operator. The range is 1 to 32; the default value is 1.
- The field delimiter character to use. The default character is the vertical bar (|).

When using the Informix DS AD Export operator, avoid using the optional Where Clause parameter to delete large tables since this can result in poor performance of your Formation job.

Also, when you load a large table using the Informix DS AD Export operator, the database may signal the error “long transaction abort.” If you receive this error, you can change the type of the input table to RAW to make it perform better so as to avoid this error, but if this table has indexes or other constraints you may want to check with your database administrator before making this change.

MSSQL Server Import Operator

The MSSQL Server Import operator reads a table in a Microsoft SQL Server database into the data flow.

To use this operator in a job, you must specify parameter values for the Microsoft SQL Server database that you want to access, the name of the database server, and the user name and password for accessing the database. In addition, you must specify the data store that describes the table structure.

The Error File parameter enables you to save in a text file any rows of data that could not be imported into the job.

MSSQL Server Export Operator

The MSSQL Server Export operator writes a data flow to a table in a Microsoft SQL Server Version 6.5 database.

To use this operator, you must specify the following:

- The table in a data store to be used by the operator.
- The database server where the Microsoft SQL Server database is located.
- The Microsoft SQL Server database to access.
- The name of the existing Microsoft SQL Server database table to which this operator writes records.
- The user name for accessing the Microsoft SQL Server database, and optionally the password.

ODBC Import Operator

The ODBC Import operator writes a table from a database that is accessed using an Intersolv 3.11 ODBC driver to a data flow. This operator requires that you have previously installed and configured the ODBC drivers that the operator uses.

To use this operator, you must specify the following:

- A table in the data store that describes the structure of a table in the database.
- The name of the table in the database that is accessed using an ODBC driver
- A valid user name for the database.
- The number of rows to use for the buffer. The range is 1 to 65,535; the default value is 500.
- The number of parallel data flows to be created by this operator. The range is 1 to 1; the default value is 1.

ODBC Export Operator

The ODBC Export operator writes a data flow to a table in a database that is accessed using an Intersolv 3.11 ODBC driver. This operator requires that you have previously installed and configured the ODBC drivers that the operator uses.

To use this operator, you must specify the following:

- A table in the data store that describes the structure of a table in the database.
- The ODBC data source to be used.
- The name of the table in the database.
- A valid user name for the database
- The number of rows to use for the extended fetch buffer. The range is 1 to 65,535; the default value is 500.

- The number of records to process before committing. The range is 0 to 2,147,483,647; the default is 0. The default value of 0 indicates that the transaction will be committed only after the whole data transfer process has completed.

If the value is set to a value greater than 0, that value should be a multiple of the value of the Rowset Size parameter. For example, if the Rowset Size parameter is set to 500, the Commit After parameter should be set to a multiple of 500, such as 1000 or 2500.

If the value is not a multiple of the value of the Rowset Size parameter, the Commit After value will be reset at run-time to be the nearest multiple greater than the value specified. For example, if the value of the Rowset Size parameter is set to 500, and the Commit After parameter is set to 800, at run-time the Commit After parameter will be set to 1000.

Oracle Server Import Operator

The Oracle Server Import operator unloads data from a table in an Oracle Server database to an input data flow. This operator uses a mix of generic and Oracle-specific parameters:

- The Where Clause parameter specifies a logical search condition to use when unloading records from the specified table.
- The Commit After parameter specifies the number of records to process before committing the transaction. This is useful for freeing limited Oracle resources.
- The Buffer Pages parameter specifies the number of buffer pages to use when importing the table.

Oracle Server Export Operator

The Oracle Server Export operator loads data from the data flow to a table in an Oracle Server database.

Red Brick Import Operator

This operator writes data from a table in a Red Brick Warehouse database to the input data flow. This operator uses the Red Brick Table Management Utility (TMU) to unload tables from the warehouse and supports all TMU UNLOAD syntax.

The Message File parameter specifies a text file where you can store the TMU messages generated by the job. Although these messages are also stored in the Red Brick Warehouse log file and the Formation log file, you may find it more convenient to store them in a separate file where you can quickly check on the work completed by the TMU.

For detailed information on TMU, please see the *Red Brick Warehouse [Table Management Utility Reference Guide](#)*.

Red Brick Export Operator

The Red Brick Export operator writes data from the data flow to a table in a Red Brick Warehouse database. The operator uses the Red Brick TMU to load tables into the warehouse, so most of the operator parameters specify how to TMU should load the table.

The following TMU functionality is not supported in this release:

- Aggregates in load mode
- EBCDIC in the discard file
- Multiple referential integrity discard files
- Per-table autorowgen

For detailed information on TMU, refer to the *Red Brick Warehouse [Table Management Utility Reference Guide](#)*.

Simple Transformation Operators

This section introduces the operators that perform simple transformations on data. These operators include:

Operator	Use this operator to . . .
Deduplicate	Remove duplicate records from the data flow.
Filter	Remove records from the data flow, based on filtering rules you specify.
Project	Add or remove columns from the data flow or alter values in a column.
Sort	Sort records in the data flow.
Union	Combine two or more non-parallel input data flows with the same format into a single output data flow.

Deduplicate Operator

The Deduplicate operator removes records with duplicate key fields from the data flow. For each set of matching records, one record from the set is sent to the Result output port. The remaining records are sent to the optional Reject output port, if there is a data flow connected to that port. The output data definition on both output ports is the same as the definition on the input port.

Common uses for this operator include the following:

- Following a union of two data flows with potentially overlapping sources
- Changing capture merge of existing and new records
- Finding unique field values prior to key generation

If the entire record is used as the key, this operator performs similarly to the DISTINCT clause in an SQL SELECT statement.

Using Deduplicate

Use the Key parameter to specify one or more fields to use when matching records. For example, you might want to deduplicate a set of records based on product code or store keys.

If the input data flow is sorted on the same fields specified in the Key parameter, specify TRUE as the value of the Sorted Input parameter. When the key fields are sorted, setting this parameter to TRUE significantly reduces the time required to perform the deduplicate operation.

Sample Job

The DedupSample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Filter Operator

The Filter operator reads records one at a time and determines whether to write the record to an output port based on filtering rules that you specify in the Filter parameter. Records that match the criteria of the filter are sent to the Valid output port. Non-matching records are sent to the Invalid output port, if there is a data flow connected to it. The output data definition on both output ports is the same as the definition on the input port.

When the Filter operator is used without defining the Invalid port, it is very similar to the WHERE clause of an SQL SELECT statement. When the Filter operator is used with the Invalid output port, it is very similar to a procedural IF statement that results in the output of TRUE and FALSE records.

Sample Job

The FilterSample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Project Operator

The Project operator reads records, one at a time, producing one output record for each input record. You can use this operator to perform the following data transformations:

- Adding or dropping columns
- Splitting a single input column into multiple output columns
- Combine multiple input columns into a single output column

Common uses for this operator include the following:

- Cleansing, formatting, and type conversions of data
- Parsing source fields into component parts
- Converting records to the required output format

The Project operator is similar to the select list portion of an SQL SELECT statement.

Sample Job

The DocSample2 job in the Samples workspace provided with the Formation product contains an example of using this operator.

Sort Operator

The Sort operator sorts records by using either a default comparison snippet based on one or more fields specified in the Key parameter or by using a comparison snippet that you define in the Compare parameter. You can set either ascending or descending sort order on each key. The records are then output in the sorted order. The output data definition is the same as the input data definition.

Use the Drop Duplicates parameter to remove records with duplicate key fields from the data flow after the records are sorted.

This operator works similarly to the ORDER BY clause in an SQL SELECT statement.

Sample Job

The SortSample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Union Operator

The Union operator combines two or more input data flows with the same format (names, data types, field order) into a single output data flow. Each time you connect an input data flow to this operator, Formation automatically adds another open port, so that you can input as many data flows as needed.

By default, the output order of the unioned data flows is Round Robin, which collects one input record from each input data flow in succession. You can choose instead to use the Preserve Sorted Order value, which interleaves the records from each input in sorted order, according to one or more fields specified in the Key parameter. If all input data flows are sorted on the same key, the output data flow will be the sorted union of all input data flows.

This operator is similar to the UNION clause in a SQL SELECT statement.



Important: *Field names are case-sensitive when attempting a union of two data flows. Therefore, you might need to propagate the data definition from one input data flow to the other or use a Project operator to transform one data flow to exactly match the other data flow.*

Sample Job

The DocSample2 job in the Samples workspace provided with the Formation product contains an example of using this operator.

Join Transformation Operators

This section describes the three join operators that you can use to join records in a data flow.

Operator	Use this operator to . . .
Join	Join data from two input sources, matching records based on comparing keys for equality.
Advanced Join	Join data from two input sources by matching keys that can be compared for equality or inequality.
Cross Product Join	Perform a nested loop join on two input sources.

Input Ports

When connecting data flows to the input ports of a join operator, you should connect the data flow with the larger number of records to the Left input port. Connect the smaller data flow to the Right input port.

Output Ports

The MatchedPair port contains all matching records. If an outer join type is specified, the output also contains unmatched records as specified by the Join Type parameter.

The MatchedLeft port contains records from the left input port that matched at least one record from the right input port. This can be used to obtain a left semi-join.

The MatchedRight port contains records from the right input port that matched at least one record on the left input port. This can be used to obtain a right semi-join.

The UnmatchedLeft port contains records from the left input port that do not match any records on the right input port. This can be used to obtain a left anti-join.

The UnmatchedRight port contains records from the right input port that do not match any records on the left input port. This can be used to obtain a right anti-join.

Join Operator

Use the Join operator to compare two input data flow tables and perform an inner equi-join as the matched output. This join operator covers most common joins, including lookups. Use the right and left matched output ports to perform validations or use the left and right unmatched output ports to find rejected records.

Using Join

To set up the join, first select the fields on which to join records. You must select one or more fields as the value of the Left Key parameter for the data coming in on the Left input port, and specify the same number, type, and order of key fields as the value of the Right Key parameter.

If both input data flows are sorted by the fields you specify in the key parameters, set the Sorted Input parameter. Sorted records boost performance in the join.

Use the MatchedPair Project parameter to specify how to process the input data fields and match them to output data fields. You can edit the data definition on the MatchedPair output port to add, remove, or modify fields coming out of this port.

Sample Job

The DocSample1 job in the Samples workspace provided with the Formation product contains an example of using this operator.

Advanced Join Operator

Use the Advanced Join operator to join data from two input sources by matching keys that can be compared for equality or inequality. Use this operator to produce different types of joins, including outerjoins, innerjoins, semi-joins, or anti-joins.

Common uses for the Advanced Join operator include the following:

- Combining or augmenting data using different sources
- Comparisons that go beyond standard key equality comparisons
- Comprehensive, combined list generation

Using Advanced Join

To set up the join, first select the fields on which to join records. You must select a Left Key value for the data coming in on the Left input port and a Right Key value for the data coming in on the Right input port.

If both input data flows are sorted, use the Sorted Input parameter. Sorted records boost performance in the join.

Use the MatchedPair Project parameter to specify how to process and match input data fields to output data fields. You can edit the data definition on the MatchedPair output port to add, remove, or modify fields coming out of this port.

The Key Match Criteria parameter uses Left and Right Keys to match records from the two input data flows, based on the equality or inequality you select. For example, you can use this parameter to match all records on the right data flow with keys less than those on the left data flow. If additional join criteria is required, use the Match parameter to specify equality or inequality matching on all fields in the input records pair for which the keys match.

Use the Join Type parameter to specify the join type of the results that will be sent to the MatchedPair Project parameter and the MatchedPair output port. You can select inner join (the default), left outerjoin, right outerjoin, or full outerjoin.



Important: When an outer join is chosen, records sent to the MatchedPair port might actually be unmatched records. These unmatched records will contain the input values of the unmatched input record and missing values for all fields that are not matched.

Cross Product Join Operator

The Cross Product Join operator compares each record on the left data flow with each record on the right data flow. If all the records match, the output is the Cartesian product of the input tables.

Common uses for this operator include:

- Combining sources with missing or ambiguous keys
- Performing list merge using free text or fuzzy comparisons

This operator is similar to the CROSS JOIN clause of an SQL SELECT statement.

Using Cross Product Join

Use the MatchedPair Project parameter to specify how to process and match the input data fields to output data fields. You can edit the data definition on the MatchedPair output port to add, remove, or modify fields coming out of this port.

Use the Join Type parameter to specify the join type of the results that will be sent to the MatchedPair Project parameter and the MatchedPair output port. You can select inner join (the default), left outerjoin, right outerjoin, or full outerjoin.



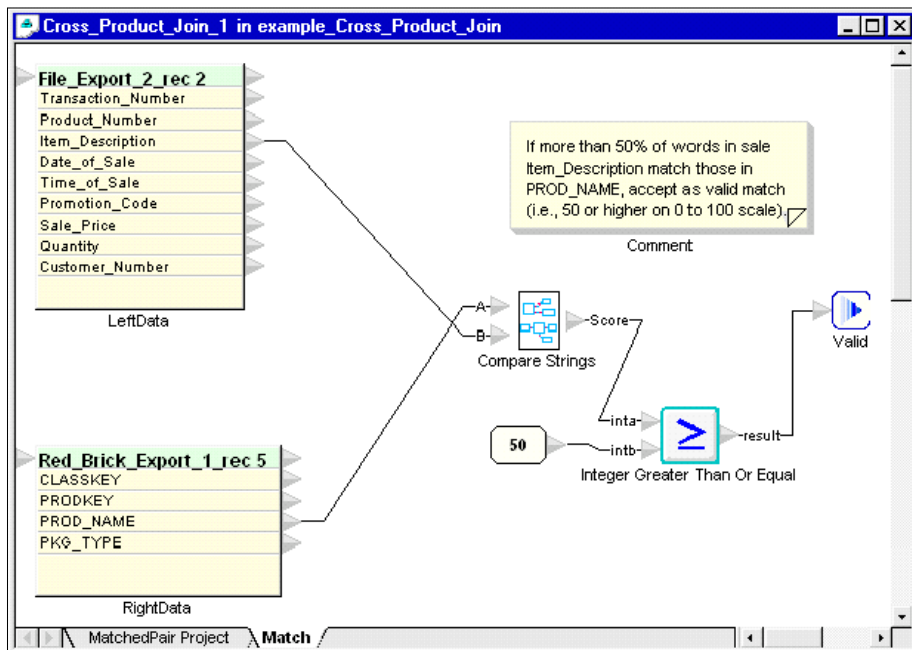
Important: When an outer join is chosen, records sent to the MatchedPair port might actually be unmatched records. These unmatched records will contain the input values of the unmatched input record and missing values for all fields that are not matched.

Use the optional Match snippet parameter to specify a function for matching left and right input records. If you do not use this parameter, all left records match all right records.

Example

The graphic below shows a Match snippet that compares input records in the Cross Product Join.

Figure 6-2
Match Snippet in a Cross Product Join



Advanced Transformation Operators

The operators in this section perform more advanced transformations specified in one or more visual snippet parameters.

To more fully understand the work of snippets in a specific operator, the Samples workspace contains jobs that illustrate the use of most of these advanced operators.

Operator	Use this operator to . . .
Aggregate	Combine records by one or more key fields and write a single record containing the summarized data to the output data flow.
Cursor	Navigate among current, previous, and next records in the data flow.
First Normal	Process records with repeating fields.
Group By	Group records by one or more fields to create one summary record for each group.
Household	Split records into groups based on a key that you specify and perform a cursor operation on each record in that group. You can also add group information to every record in the group.
Program	Integrate independent executables into a job.
Split	Separate the input data flow into multiple output data flows.
Surrogate Key	Adds generated artificial keys, known as surrogate keys, to records.

Aggregate Operator

Use the Aggregate operator to combine records by one or more key fields and write a single record containing the summarized data to the output data flow. The Aggregate operator is a simpler version of the Group By operator that summarizes all records rather than summarizing by a user-defined grouping key. This operator is similar to the `SELECT MIN(column)` or `SELECT MAX(column)` statement in SQL.

Common uses for this operator include the following:

- Finding minimum/maximum field values
- Collecting parameters as input to subsequent jobs
- Creating counts and statistics to confirm job completeness

Using Aggregate

To use the Aggregate operator, first define a temporary record where the summarized data will be stored during the aggregation process. This temporary record, called the Summary Record parameter, defines the fields and data types that will be aggregated.

After defining the Summary Record parameter, define the aggregation process using the following parameters:

- The Initialize parameter defines the starting values for each field of the summary record.
- The Aggregate parameter specifies how to process the input records and store the resulting values in the summary record. For example, the aggregation can take the form of adding fields or finding a minimum or maximum value for a field.
- The Finalize parameter maps how the summary record is to be written to the output data flow after all the records in the input data flow have been processed.

Sample Job

The AggregateSample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Cursor Operator

The Cursor operator scans a data flow using a “window” of the current, previous, and next records in the data flow and applies a user-supplied rule to every record. This operator also provides the position of the current input record in relation to the set of input records. You can use the Cursor operator to perform the following actions:

- Filtering that discards any subset of input records
- Mapping that uses projection to transform each input record into an output record
- Combining values from consecutive records related in some way

Common uses for this operator include the following:

- Identifying units or transactions in sequenced data
- Fixing irregularities prior to Sort, Join, Group By
- Assigning sequential keys

Using Cursor

The Cursor snippet parameter specifies how to process the input record. The snippet shows the previous, current, and next records in the data flow so that you can compare and use fields from any of these three records to examine the data.

First Normal Operator

Use the First Normal operator to process records with repeating fields; for example, to break an input record into one or more detail records. This operator is similar to the Project operator, but allows multiple output records to replace a single input record.

Common uses for this operator include the following:

- Building unique records for each value in a repeating group
- Creating separate header and line item records

The First Normal operator provides two output ports:

- The Detail port contains one record for each occurrence of the repeating field. A connection to this port is required.
- The Master port contains the fields that apply to all the detail records that are output for a single input record. A connection to this port is optional.

Using First Normal

The Decompose Relations parameter is a visual snippet in which you specify the fields to use in denormalizing the input records. This is a required parameter. By default, the snippet by contains a Repeat function that you use to set up the programming logic.

Sample Job

The FirstNormalSample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Group By Operator

Use the Group By operator to group records by one or more fields, then aggregate the data in each group, and write the summarized data for each group to the output data flow. This operator is similar to the GROUP BY clause in an SQL SELECT statement.

Common uses for the Group By operator include the following:

- Combining, cleaning, and packaging low-level data
- Creating composite fact tables
- Creating dimension tables

Using Group By

The Key parameter specifies the fields to use when grouping the input records.

The Summary Record parameter defines a temporary record in which the summarized data will be stored during the aggregation process.

Use the Initialize, Map, Aggregate, and Finalize snippet parameters to specify the following actions the operator should take when processing each group.

- Whenever the operator starts processing a new group, the Initialize parameter sets the initial values for a “bucket” for the group and also initializes any temporary variables used in the Aggregate snippet.

- The Map snippet defines how the fields in each record are processed and added to the summary record for the group.
- The Aggregate snippet represents the process of summary records being added together. Both summary record “buckets” represent one or more records in the group, while the SummaryRecordResult indicates the final disposition of all records in the group.
- When all the input records for the group have been processed, the summary record is written to the results output data flow, as specified by the Finalize parameter.

If the input data flow contains records that are already sorted by the fields you specify in the Key parameter, you can optimize the processing of these records by setting the Sorted Input parameter to True.

Sample Job

The DocSample1 job in the Samples workspace provided with the Formation product contains an example of using this operator.

Household Operator

The Household operator first divides records into groups based on a key that you specify, then it uses the Cursor parameter to process the records in each group.

The Household operator is similar to the Cursor operator, but the Household groups records prior to executing the Cursor parameter, so that the Cursor parameter operates only on records within a group.

Common uses for this operator include the following:

- Grouping units or transactions and augmenting data
- Assigning keys or common identifiers each record in a group
- Direct marketing-style household classification

Using Household

Use the Key parameter to specify the fields to use when collecting records into groups; note that these values are only compared for equality. To compare key fields for inequality, use the Compare parameter.

If the input data is already sorted by the key field, set the Sorted Input parameter to TRUE for more efficient record processing.

Use the Cursor parameter to specify how to process input records.

Use operator variables to hold condition, count or other reference values within and across groups.

Sample Job

The HouseholdSample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Program Operator

The Program operator enables you to integrate independent executable applications into a data flow diagram. The Program operator uses different ports than other operators, as shown in the following table.

Port Name	Type	Is attached to. . .
Stdin (optional)	Input	The data flow containing the input data flow for the operator, if one exists.
Stdout (optional)	Output	The data flow containing the output data flow for the operator, if one exists.
Stderr (optional)	Output	The data flow containing the error log output data flow for the operator, if one exists.

Because the purpose of this operator is to call an outside program, the Executable parameter that supplies information about the executable is required. Any arguments required by the executable must be specified in the Arguments parameter.

You must also specify the format of data sent to each port that you use. For example, if you connect a data flow to the Stdout port, you must specify a value for the Stdout Record Def parameter.

The following sections show sample uses of the Program operator.

Example of Program Operator Calling the awk Command

In this example, the user wants to execute the following UNIX shell command from within a Formation job using the Program operator:

```
awk 'BEGIN {FS=":"} {if ($3!=$4&&($3<10||$4<10)) print $1"|" $5)'/etc/passwd
```

To execute this command using the Program operator, the Executable parameter would have the value `awk` and the Arguments parameter would have the following value:

```
'BEGIN {FS=":"} {if ($3!=$4&&($3<10||$4<10)) print $1"|" $5)'/etc/passwd
```

Important: The values for the argument parameters can include embedded spaces.

In this example, the input file contains the following text:

```
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
sys:x:3:3:/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
smtp:x:0:0:Mail Daemon User:/:
listen:x:37:4:Network Admin:/usr/net/nls:
```

After the Program operator has processed the input file using the `awk` command, the resulting output data flow would contain the following text:

```
root|Super-User
lp|Line Printer Admin
listen|Network Admin
```

Example of Program Operator Calling the find Command

In this example, the user wants to execute the following UNIX shell command from within a Formation job using the Program operator:

```
find /etc \( -name '*.conf' -o -name '*.rc' \) -print
```





To execute this command using the Program operator, the Executable parameter would have the value `find` and the Arguments parameter would have the following value:

```
/etc
(
  -name
  '*.conf'
  -o
  -name
  '*.rc'
)
-print
```

Important: The shell command has escape characters; the argument list does not.

After the Program operator has issued the `find` command, the resulting output data flow would contain the following text:

```
/etc/inet/inetd.conf
/etc/inetd.conf
/etc/mail/mailx.rc
/etc/mail/Mail.rc
/etc/nscd.conf
/etc/rpld.conf
/etc/nfssec.conf
/etc/pam.conf
/etc/nsswitch.conf
/etc/syslog.conf
/etc/vold.conf
/etc/rmmount.conf
/etc/fn/x500.conf
/etc/printers.conf
:
```

Split Operator

The Split operator separates the input data flow into multiple output data flows, such as nonoverlapping sets split by logic that you specify in the Distribution snippet parameter. Each input record is assigned to one of the output data flows, based on the default hash value or a user-defined hash value. The output data definition is the same as the input data definition.

Common uses for this operator include the following:

- Create a certain number of populations for different processing
- Select control sample for statistical analysis

- Classify record types for variant processing

Using Split

Each time that you connect an output data flow to the Split operator, Formation automatically adds another open port so you can split the data into as many output data flows as needed.

Use the Hash parameter to specify programming logic that examines the values in the fields specified in the Key parameter. For each input record, the Hash snippet parameter returns an integer between zero and the number of output data flows that determines where the output port where the record will be directed.

Sample Job

The SplitSample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Surrogate Key Operator

You use the Surrogate Key operator to add generated artificial keys, known as surrogate keys, to records. The Surrogate Key operator requires that both input ports and one or more output ports be used.

To use this operator, you must specify the following:

- The natural key for the records at the Data port. This is the natural key to be assigned a surrogate key. The fields in the Data Natural Key are mapped to those in the Surrogates Natural Key using the Data to Surrogates Natural Key Project snippet.
- The natural key for the records at the Surrogates port. This is the natural key that has already been assigned a surrogate key. The fields in the Data Natural Key are mapped to those in the Surrogates Natural Key using the Data to Surrogates Natural Key Project snippet.
- The field in the records at the Surrogates port that contain the Surrogate Key. This field must have an Integer data type and must not be used as part of the Surrogates Natural Key.

- A snippet that maps the fields in the Data Natural Key to those in the Surrogates Natural Key. The Data Key data block lists the one or more fields in the Data Natural Key. The Mapping Key result block lists the one or more fields in the Surrogates Natural Key.
- Total megabytes of memory available to the hash algorithm. Range is 1MB to 2GB, inclusive. Default is 16MB.

Sample Job

The SurrogateKeySample job in the Samples workspace provided with the Formation product contains an example of using this operator.

Data Parallelism Operators

Although you can use the Parallel parameter in some operators to create data parallelism, you can also use the Partition and Gather operators to explicitly control parallelism in a data flow.

Operator	Use this operator to . . .
Partition	Divide a single input data flow into multiple parallel output data flows.
Gather	Collect individual parallel data flows and combine them into a single output data flow.

Partition Operator

Use the Partition operator to divide a single input data flow into multiple parallel output data flows for parallel processing of data. Data flows separated by the Partition operator can be combined by the Gather operator.

When you add a Partition operator to a data flow diagram, you can connect only a single output data flow to the Partition operator. This single data flow maps to multiple output data flows. Logically, there is a single output data flow even though physically there are multiple output data flows.

Using Partition

The default number of parallel data flows is two, but you can set the `Parallel` parameter to override the default value.

The `Distribution` parameter controls the manner in which data is placed in the output data flows:

- The Round robin value (the default) distributes the records evenly among the output flows.
- The Broadcast value sends every record to every output flow.
- The Hash value specifies the programming logic that examines the values in the fields specified in the `Key` parameter. For each input record, the `Hash snippet` parameter returns an integer between zero and the number of output dataflows that determines where the output port where the record will be directed.

Gather Operator

Use the Gather operator to collect individual parallel data flows and combine them into a single output data flow. Use this operator to combine parallel data flows that were created by the Partition operator or by a File Import operator.

Using Gather

The default number of parallel data flows is two, but you can set the `Parallel` parameter to override the default value.

The `Output Order` parameter specifies how to send the input data flows to the output data flow:

- The Round robin value (the default) gathers the input records one at a time from each input data flow.
- The Preserve sorted order value interleaves the records from each input dataflow in sorted order into the single output dataflow. Use this value only if the input is sorted by the fields you specify in the `Key` parameter.

Performance Considerations

The following are guidelines to improve the performance of a data flow diagram:

- Minimize the amount of data each operator must process. For example, if some fields in the input data flow are never used in the job, use a Project operator to drop those fields immediately after importing the data. Operators that remove data include Filter, Project, Group By, Deduplicate, Household, and Sort.

It is especially important to try to minimize the amount of data processed by an Advanced Join, Join, or Cross Product Join operator.

- When possible, sort the data flows that are input into an Advanced Join, Join, or Cross Product Join operator to speed their processing, and set the Sorted Input parameter to those operators to Sorted Ascending or Sorted Descending, whichever is appropriate.
- When possible, avoid creating data flow diagrams that take a single data flow, separates it into multiple data flows, and then combines those data flows using the Advanced Join, Cross Product Join, Gather, Join, or Union operators.

If multiple data flows going into a Union operator are not sorted, you should not sort them before the Union because the performance loss from the sort operation usually is not offset by the performance gain of sorted input to the union operation.

Working with Snippets

In This Chapter	7-3
Overview of Visual Snippets	7-3
Defining a Visual Snippet	7-5
Variables	7-6
Ordering Links	7-6
Limitations	7-7
Standard Functions.	7-7
Syntax Functions	7-8
The Comment Function	7-8
The Package Function	7-9
The If Then Else Function	7-10
Custom Functions	7-14
Operator Variables	7-15
Defining an Operator Variable	7-15
Limitations	7-16
Data Types Used in Visual Snippets	7-16
The Null Value	7-17
Using Null Values with Import or Export Operators.	7-18
Using Null Values in Operators with Key Parameters	7-18
Using Null Values in Visual Snippets	7-19
Constants and Lists	7-19
Creating a Constant	7-20
Type Conversions	7-20
Automatic Conversion	7-20
Manual Conversion	7-21
Code Snippets	7-21

In This Chapter

A snippet allows you to specify the programming logic for an operator parameter. You can specify a transformation as simple as dropping columns or as complex as comparing the same field in three adjoining records in the input data flow and sending the highest value to a field in the output data flow.

A visual snippet enables you to visually specify the programming logic for an operator parameter, and a code snippet allows you to enter the programming logic as C++ source code. In general, use visual snippets to specify your operator parameters.

This chapter explains the concepts involved with using snippets in Informix Formation. Sections in this chapter include the following:

- [Overview of Visual Snippets](#)
- [Defining a Visual Snippet](#)
- [Standard Functions](#)
- [Custom Functions](#)
- [Operator Variables](#)
- [Data Types Used in Visual Snippets](#)
- [Code Snippets](#)

Overview of Visual Snippets

A visual snippet enables you to visually specify an operator parameter by dragging and dropping standard functions, custom functions, and operator variables into a window where these elements are combined with the input and output data flows for the operator.

The concept of a visual snippet is that it is easier to visually draw a transformation than it is to code it in a programming language. This visual view of the transformation enables you to spot errors in logic more easily than reading through pages of code and it also provides a self-documenting roadmap of its own functionality to anyone learning the purpose of the snippet and its place in the job. For example, the snippet in the following illustration specifies an AND condition that compares both Date and Dollars values to constant values. You can follow the logic by tracing the data flow through the snippet components.

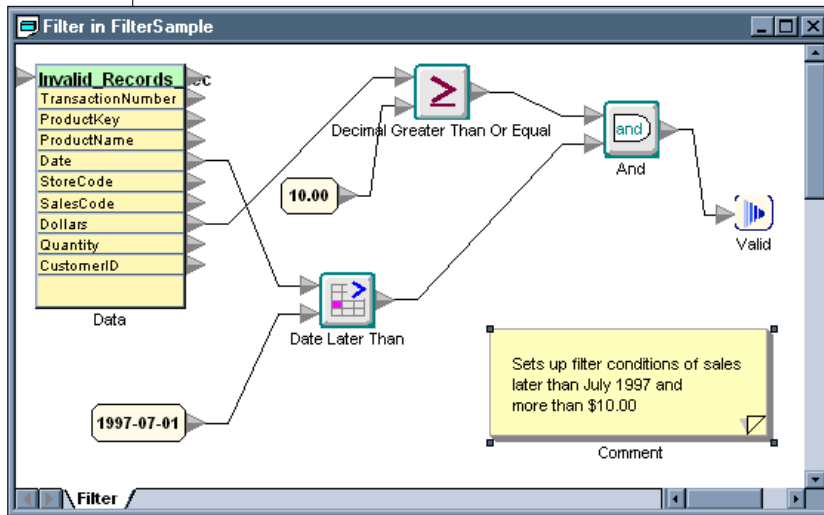


Figure 7-1
*Filter Snippet
Comparing Date and
Dollars to Constants*

When creating visual snippets, use the following conventions:

- Do not create any variables or constants with the name of NULL, TRUE, FALSE, or any names reserved for use by the C++ language.
- Compilation errors may indicate that there is a naming conflict in the code generated by Formation. To correct the naming conflict, check that the names of any variables you create in the affected section of code are unique, then regenerate the code and try the compilation again.

Defining a Visual Snippet

When you open a visual snippet window, you usually see one or more yellow record definitions as a starting point to the work you want the snippet to perform. These record definitions represent the format of the data coming into the snippet (usually named the Data block) and the format of the data exiting the snippet (usually named the Result block).

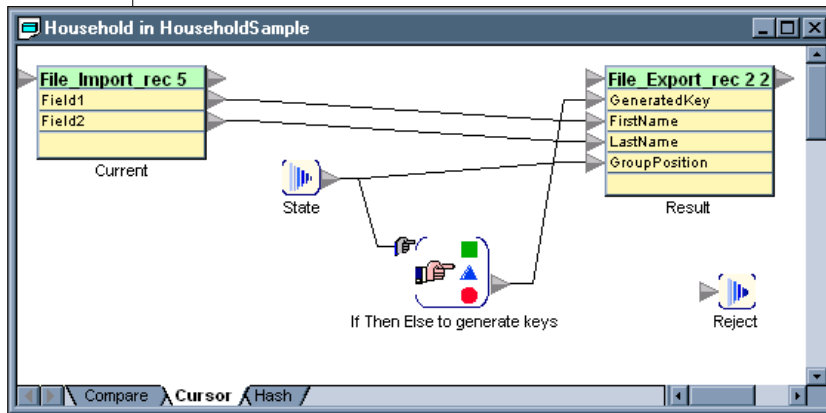


Figure 7-2
Data Blocks and
Result Blocks



Important: If you do not see the Data or Result blocks, check that you have connected the appropriate data flows to the operator using the snippet.

Inside a visual snippet, you can use any of the following components:

- Standard functions. Refer to [“Standard Functions” on page 7-7](#) for more information. The previous graphic shows a single standard function, If Then Else.
- Custom functions. Refer to [“Custom Functions” on page 7-14](#) for more information.
- Operator variables. Refer to [“Operator Variables” on page 7-15](#) for more information.
- External libraries. Refer to [Chapter 10, “Advanced Job Topics,”](#) for more information.
- Variables. Refer to [“Variables” on page 7-6](#) for more information. The previous graphic shows two variables, State and Result.

- Ordering links. Refer to [“Ordering Links” on page 7-6](#) for more information.

Variables

A snippet can contain one or more variables that provide standard functionality, such as passing the state of a field or record back to the operator. For example, the Filter snippet in the Filter operator uses a Valid variable to determine whether the current record meets the filter conditions specified in the snippet.

Variables use input or output ports, similar to operator input or output ports.

When using a variable, you must pay attention to the data type of the variable. For example, the Valid variable in the Filter snippet requires that input to it be a logical data type. To see the data types of variable ports, place the cursor on the port and check the status bar for the type.

The If Then Else function is an efficient way to handle conditions in a variable. For example, if the State variable can output values of 0, 1, 2, or 3, you can set up the If Then Else function with case tabs for all four of those values.

Ordering Links

Usually, the visual flow of data in a snippet is ordered correctly by default because there is no required execution order. However, for very complex snippets that depend on processing data in a specific order, you might need to specify ordering links in the following cases:

- When operator variables that are used several times in the snippet hold values that must be set or reset in sequence.
- When one or more external functions must access an external state at a specific point in the snippet.
- When you want to use the Write Message to Log Utility function at specific points in a snippet.

To display ordering links, select Ordering Links on the View menu or select Show Ordering Links on the context menu while in the visual snippet window.

Limitations

You can set ordering links only within a single snippet level. Also, you cannot use control and data flow loops.

Standard Functions

Informix Formation provides an extensive set of standard functions that you can use to perform various tasks in a visual snippet. The Standard Functions section of the Toolbox tab in the Navigator lists the different kinds of functions, as in the following list.

- Numeric functions (Decimal, Double, Float, Integer, Unsigned Integer). These functions include common mathematical, comparison, and other calculations.
- Text functions, such as Search and Split, Compare, and Text Trim. These functions provide text parsing and manipulation.
- Logical functions: And, Not, and Or.
- Time-related functions, subdivided into Date, Time, Timestamp, Interval functions.
- Syntax functions, including Comment, Package, and If Then Else functions. Refer to [“Syntax Functions” on page 7-8](#) for more information.
- System functions, which include Is NULL, Sequential ID, and Write Message to Log Utility.
- List functions, which include functions to find an element in a list and the length of a list.
- Conversion functions. Refer to [“Type Conversions” on page 7-20](#) for more information.

The Informix Formation online help describes each standard function in detail. When you are using a function, press the F1 key to display the online help topic for that function.

Syntax Functions

The syntax functions provide handy shortcuts when working in visual snippets. The most commonly used syntax functions are:

- [The Comment Function](#)
- [The Package Function](#)
- [The If Then Else Function](#)

The Comment Function

Create a comment whenever you want to place explanatory text in a job window or in a visual snippet. For example, if you want to note that a visual snippet uses an operator variable with an initial value of 100, you can create a comment that stores that information and displays it whenever the snippet window is open.

To create a comment, select New Comment on the context menu in a job window or in a snippet window or drag the Comment icon (in the Syntax folder of the Standard Functions folder) and drop it in a job window or snippet window.

The following graphic shows a visual snippet that uses comments as documentation for the snippet logic.

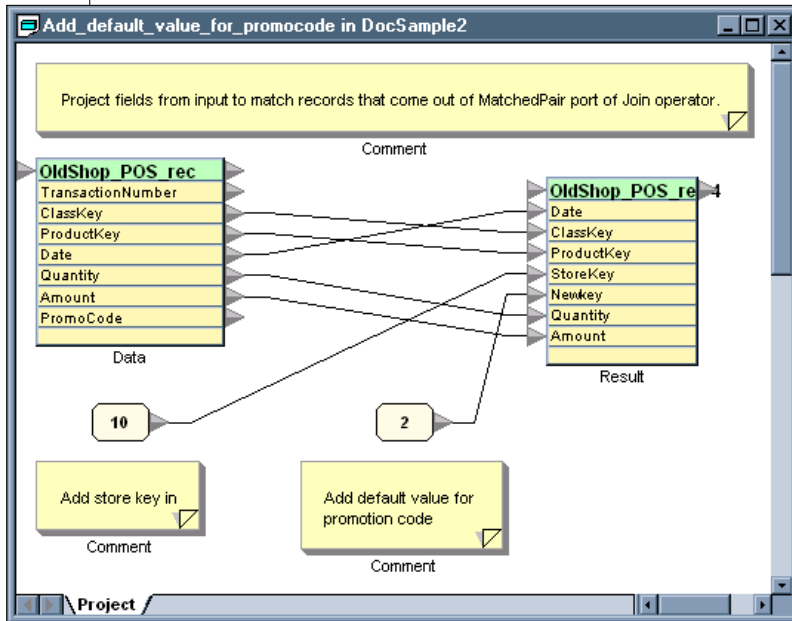


Figure 7-3
Comments in a Visual Snippet

The Package Function

Use the Package function to visually “hide” the complexity of snippet logic. For example, a snippet with five or six complex calculations can be easier to read if some or all of the complex calculations are stored in packages.

Setting Up a Package

To create a package, select the items in the snippet you want to be in the package and select Replace with Package on the Edit menu.

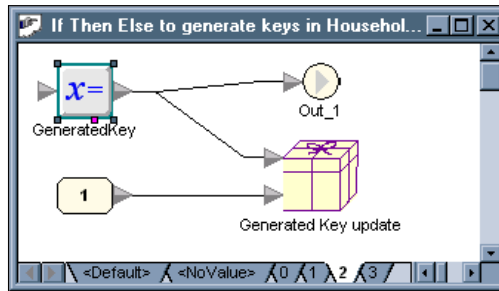


Figure 7-4
Creating a Package

To remove a package, select the package and select Replace with Contents on the Edit menu.

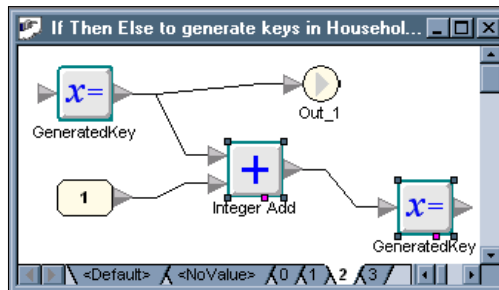


Figure 7-5
Removing a Package

The If Then Else Function

Use the If Then Else function to provide different logic for different values of a variable, field, or other component of a visual snippet. If necessary, you can nest If Then Else functions to handle complex conditional logic.

In the following graphic, the snippet uses the If Then Else function to handle all possible values for the State variable.

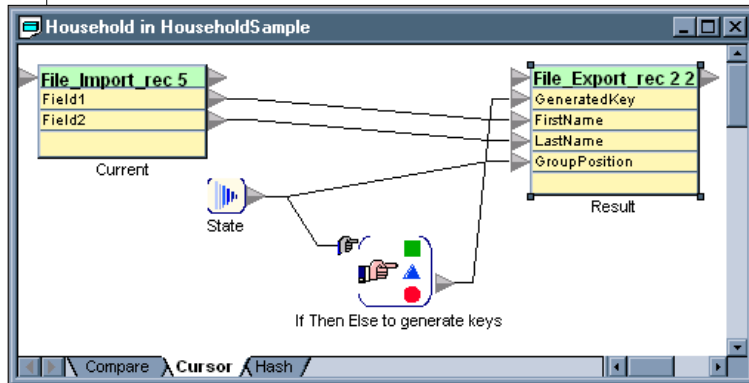


Figure 7-6
Using If Then Else

Because the State variable in this snippet can have a value of 0, 1, 2, or 3, the If Then Else function must include logic to handle all those values. Use the Default tab to specify values that do not require special handling and use the No Value tab to handle Null values.

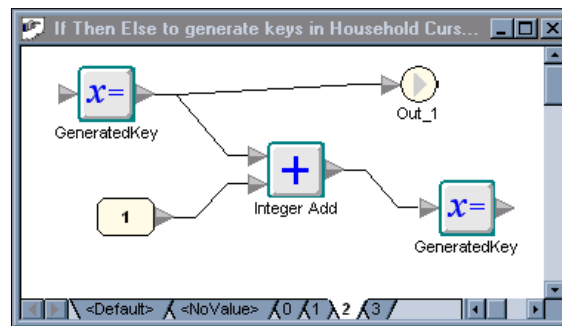


Figure 7-7
The Default Tab

Setting up the If Then Else Function

To use the If Then Else function, you must perform the following tasks. Note that the text references the graphics on the previous page to illustrate the task being discussed.

- Add the If Then Else function to the snippet.

You can either drag and drop the function (located in the Syntax folder under Standard Functions on the Toolbox tab) or use the New If Then Else command on the snippet window's context menu.

- Connect the value you want to branch on to the Selection port of the If Then Else function.

In the sample snippet, the State variable is connected to the Selection port.

- To bring other values into the If Then Else function or output values from the function, create input and output ports for the function, as needed. Each of these ports will then be displayed on each tab of the function as input or output variables. To add a port, select the If Then Else function and choose New Input Port or New Output Port on the context menu.

In the sample snippet, the function has one output port defined.

- Connect fields to the input and output ports, as needed.

In the sample snippet, the output port on the function is connected to the GeneratedKey field on the Result data block, so the result of the If Then Else function will be stored in that field.

- Define the choices available in the function. Each choice will be displayed as a separate tab in the snippet window.

To define the choices, select Edit Choices on the context menu while the function is selected. To add a choice, type the choice in the Choices field and click Add Choice. You cannot delete the Default and No Value choices.

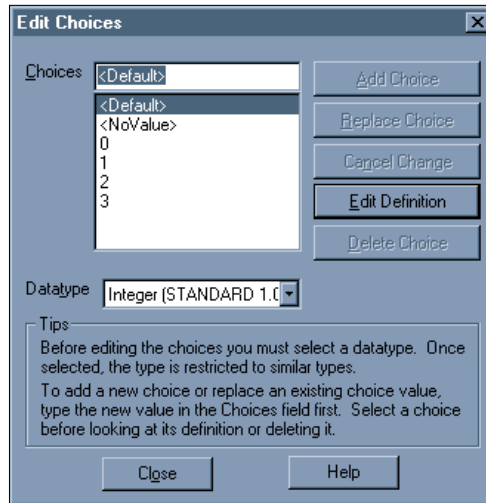


Figure 7-8
*The Edit Choices
Dialog*

- Provide the logic for each tab in the function. The following graphic shows the logic in the sample snippet for the values of Default and 2. In order for the 2 tab to correctly process the data, you should use ordering links to specify that the key goes to the output variable before the addition occurs.

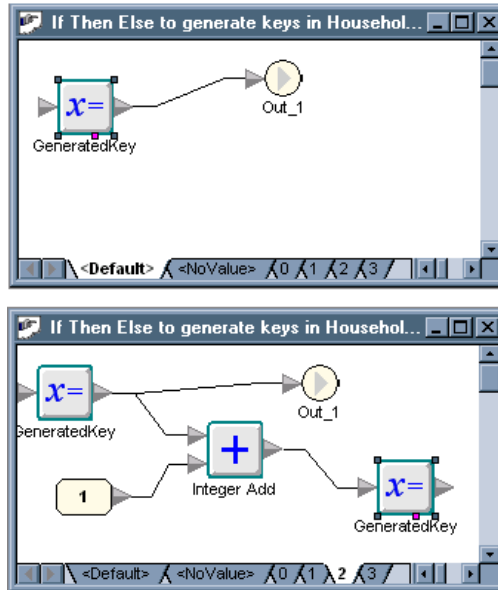


Figure 7-9
The Logic Behind
the Default and 2
Values

Custom Functions

You can create custom functions that act as subroutines within the workspace. These custom functions can be shared among snippets and jobs in the same workspace. Any functions you create are stored in the Custom folder on the Toolbox tab of the Navigator.

To create a custom function, you can either define it from scratch or select sections of a visual snippet and execute the Replace with Function command on the Edit menu. After you have created the visual function, you can use it the same as any of the standard functions. For example, you can nest custom functions.

Operator Variables

An operator variable provides temporary storage for information that you want to use across records and snippets inside a single operator. For example, you can use an operator variable to store a surrogate key value that is generated for each group of records or to store running aggregate values.

The definitions for the operator variables you have defined in a workspace are shown in the Custom Functions folder of the Toolbox tab of the Navigator. Before the operator processes any data, it fetches the initial value for the operator variable from the definition. Whenever the value of the operator variable changes while the operator is processing data, that new value is available immediately for use inside the snippet or elsewhere in that operator.

Defining an Operator Variable

To create a new operator variable, select New->Operator Variable on the File menu. This opens a dialog where you can define the fields in the operator variable, including the field names, initial values, data types, and descriptions.

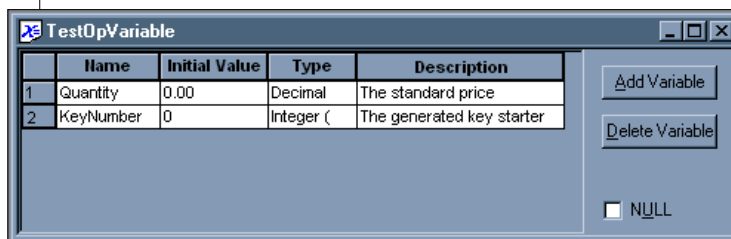


Figure 7-10
The Operator Variable Dialog

You can initialize the fields in an operator variable in any of the following three ways:

- The Operator Variable dialog box
- The Constructor snippet in the operator in which the variable is used
- Inside the snippet itself, based on logic that you provide

To use the operator variable in a visual snippet, drag it from the Toolbox tab of the Navigator to the visual snippet window. You can then use it in the snippet by adding it to the appropriate place in the data flow inside the snippet.

Limitations

You cannot pass the value of an operator variable from one operator to another operator.

Operator variables are not shared among parallel instances of an operator. For example, if you are using an operator variable to generate keys, do not use parallel data flows because the current value of the key stored in the operator variable is not shared among the data flows.

Data Types Used in Visual Snippets

The following table lists the data types available inside a visual snippet. Note that these data types are a subset of the Formation data types that you can see when editing record definitions for a data store or for an operator port.

Data Type	Format	Description
Text		Variable-length string
Integer		Signed 4-byte number
Unsigned Integer		Unsigned 4-byte number
Float	1.23456E+006	Floating decimal point; precision 1 to 6 digits
Double	1.23456789012345E+015	Floating decimal point; precision 1 to 15 digits
Decimal		Fixed decimal point; precision 1 to 38 digits; scale 0 to precision value (digits after decimal)
Logical		Boolean True (1) or False (0)

(1 of 2)



Data Type	Format	Description
Date	yyyy-mm-dd	Fixed precision in years, months, days
Time	hh:mm:ss. uuuuuu	Precision in hours, minutes, seconds, milliseconds
Timestamp	yyyy-mm-dd hh:mm:ss. uuuuuu	Precision in years, months, days, hours, minutes, seconds, milliseconds
Interval	dd:hh:mm:ss.uu uuuu	Unsigned delta time; precision in days, hours, minutes, seconds, milliseconds

(2 of 2)

Tip: A constant can be defined as a list of these types; for example, a list of text strings.

The Null Value

Null values are introduced into a Formation job in one of the following ways:

- Data that is imported into Formation contains Null values, for example, a field separated text file with no data between two separators would indicate a field with a Null value.
- The user explicitly assigns Null values in a visual snippet.

All Formation data types can handle Null values. Therefore, common functions can return Null, as shown in the following examples:

```
X + Y or X * Y or X / Y when Y is Null
X Greater Than Y when Y is Null (logical)
Text Concatenate A, B when B is Null
Text Compare A, B when B is Null (logical)
AND logicals P, Q when Q is Null
```

Using Null Values with Import or Export Operators

When using import and export operators, be aware of how those operators handle fields with Null values. When importing records, empty fields are imported as Null values. When exporting records, you must explicitly set fields to Null.

Using Null Values in Operators with Key Parameters

Null values in key parameters are handled differently by different operators as shown in the following table.

Operator	Use of Null Values
Deduplicate	Treats Null as a distinct value. Only one record is output for all input records containing the same Null key fields.
Sort	Treats Null as a distinct value. Null is treated as the smallest possible value. If the Drop Duplicates parameter is set to a value of True, only one record is output for all input records containing the same Null key fields.
Group By Household Surrogate Key	Treats Null as a distinct value. All records with the same key field s are grouped together.
Join Advanced Join Cross Product Join	Treats Null as a special value that does not match any other value, including other Null values.
Partition Split	When the Distribution parameter is set to a value of Hash, these operator treats Null as a distinct value. Only one record is output for all input records containing the same Null key fields.
Gather Union	When the Output Order parameter is set to a value of Preserve sorted order, these operators treat Null as a distinct value. Null is treated as the smallest possible value. Only one record is output for all input records containing the same Null key fields.

Using Null Values in Visual Snippets

Inside snippets, you might need to provide explicit handling for Null values because a Null result is possible for all calculations and logic. You can use the Is Null function to test for Null values or you can use the If Then Else function to handle a Null value in a field.

If a snippet is accumulating values in a field and one of the values in that field is a Null, the field value changes to Null until the field is explicitly reset. Keep this in mind when performing operations such as count, sum, and average.

A constant of " " or <NoValue> does not compare to Null.

Constants and Lists

You can create constants of any data type, including lists of values of any data type. For example, you can create a list of text strings that are used as input values to the Text Search in List function, as shown in the following graphic. Or you can use a single-value constant as a comparison to an Equal function. Constants are static, so you cannot modify the assigned values at runtime.

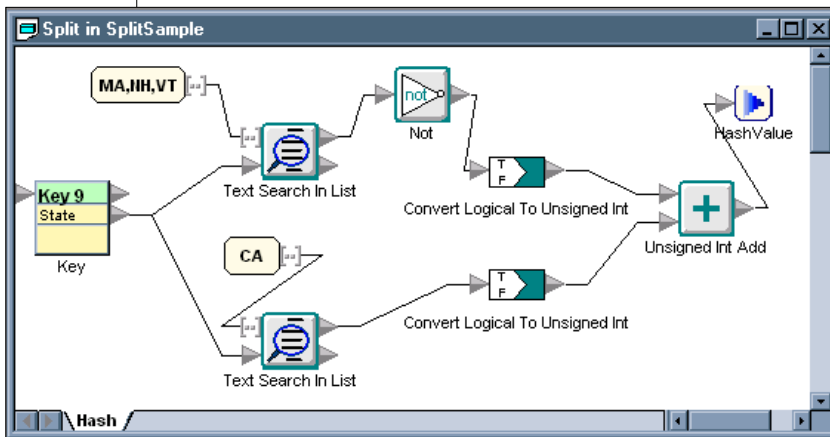


Figure 7-11
List of Text
Constants in a
Visual Snippet

Creating a Constant

To create a constant, select the New Constant command on the context menu in a visual snippet. You can either type a value directly into the constant or use the Define. . . command on the context menu while the item is selected.

To change the data type of a constant, select the constant and then choose Define. . . from the context menu to display a dialog in which you can edit the data type and value. To create a list constant, choose List of *<datatype>* as the data type and then type in the values for the list items.

Type Conversions

Formation can handle type conversions inside visual snippets either automatically or manually as described in the following sections.

The Conversion folder under the Standard Functions item on the Toolbox tab of the Navigator lists all the conversion functions available. The online help contains descriptions of each of these functions.

Automatic Conversion

Automatic conversion occurs when there is only one way to convert the data type; for example, when converting an integer to a float. A circle on the line between two data types indicates an automatic conversion. To see the conversion being used, point the cursor at the line and then check the status bar at the bottom of the Architect window.

While automatic conversions can be useful, you should check the conversion when using Text, Date, Time, or Timestamp data types so that the converted data type is the correct one for the snippet.

Manual Conversion

You must manually specify the type conversion when the software cannot automatically convert one data type to another. When automatic conversion is not available, you cannot connect the two data types without placing a conversion function between them. For example, if you try to connect a float to an integer, the software knows that the float value can either be rounded or truncated, so you must manually select the conversion that you want to occur.

Code Snippets

In rare cases, you might choose to use a code snippet in place of a visual snippet. In a code snippet, you write C++ code that will be inserted into the job code when that code is generated.

You can view any visual snippet as a code snippet by selecting Code Editor on the View menu when a visual snippet window is active.

If you modify code generated by a visual snippet, you must remove the comment "// Formation Generated" from the top of the code snippet. If you do not remove this line, the C++ syntax you have entered will be lost the next time that parameter is viewed as a visual snippet and will be replaced by C++ syntax automatically generated by the visual snippet.

When creating a code snippet, use the following guidelines to ensure that the snippet will work correctly within the Formation software:

- Do not throw exceptions.
- Free any memory that you allocate.
- Allocate and free memory using the new and delete C++ functions. Do not use the malloc and free functions.
- When creating and using global and temporary variables, all C++ data types are valid.
- When manipulating existing data within a function, all C++ data types are valid, as well as blob and string. blob is implemented as an array of unsigned characters. string is the same as the C++ char* data type and so you can use all string manipulation functions, such as strcpy and strcmp.

- Do not create and use static variables.
- Do not create functions that cannot be processed in a threaded environment.

Creating and Modifying Flow Engine Machines

In This Chapter	8-3
Introduction to Flow Engine Machines	8-4
Creating or Modifying a Flow Engine Machine	8-5
Defining Temporary Space for Job Execution	8-8
Using the Temporary Space Dialog	8-9
Default Temporary Space Locations.	8-10
Guidelines for Setting Temporary Space	8-11

In This Chapter

This chapter describes how to specify the computer, directory, and other information you need to provide as part of the computing environment in which a job will run. This specification is called a *Flow Engine machine*.

This chapter is organized into the following sections:

- [Introduction to Flow Engine Machines](#)
- [Creating or Modifying a Flow Engine Machine](#)
- [Defining Temporary Space for Job Execution](#)

Introduction to Flow Engine Machines

When you generate the source files for your job and build them into executables, you specify a Flow Engine machine that defines where those files are created and built. By default, the computer is the local Windows NT system on which the Formation Architect component is installed. This default computer specification appears in the Generate Job Files dialog in the Generate for flow engine machine field as *** local machine ***, as shown in the following illustration.

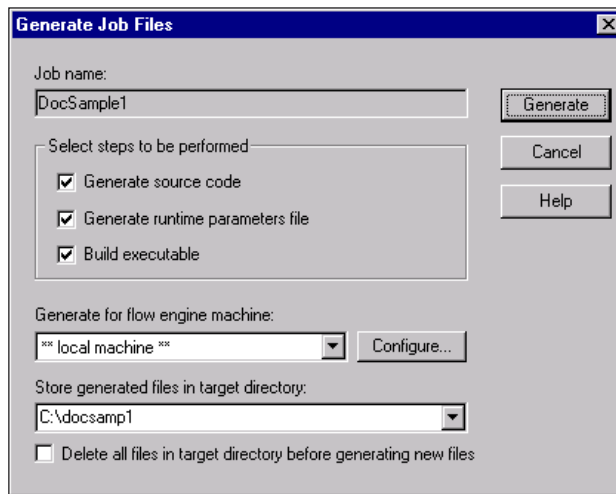


Figure 8-1
*The Generate Job
Files Dialog*

To build the job executable files on a computer other than the one on which the Formation Architect component is installed, do the following:

1. Select a remote computer on which to run the job that has the Flow Engine component installed.
2. Configure that computer as a Flow Engine machine that you can select from the Architect component. For more information on configuring a computer as a Flow Engine machine, refer to [“Creating or Modifying a Flow Engine Machine”](#) on page 8-5.
3. Select the newly configured Flow Engine machine from the Generate Job Files dialog. For more information on using the Generate Job Files dialog to generate the files for the job, refer to [Chapter 9, “Generating and Running a Job.”](#)

The following sections describe how to use the various dialogs to create, modify or delete a Flow Engine machine.

Creating or Modifying a Flow Engine Machine

You use the Flow Engine Machines dialog to create, modify or delete Flow Engine machines. You select a Flow Engine machine to use when you generate and build the files for a job. You access the Flow Engine Machines dialog by doing either of the following:

- By selecting Configure Flow Engine Machines from the Tools menu.
- By clicking the Configure button on the Generate Job Files dialog.

The following illustration shows the Flow Engine Machines dialog.

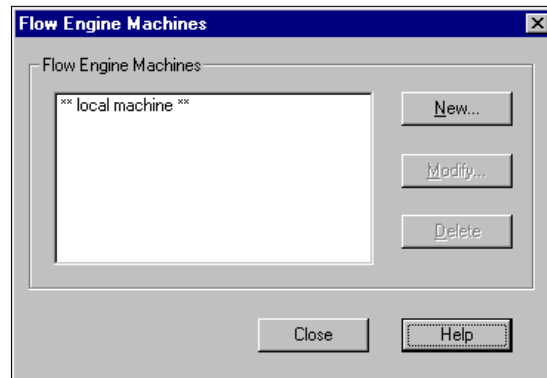


Figure 8-2
*The Flow Engine
Machines Dialog*

The following table lists the tasks you perform using the Flow Engine Machines dialog and how to perform those tasks.

To...	...Do the Following
Create a Flow Engine machine	Click the New button. The Configure Flow Engine Machine dialog is displayed.
Modify a Flow Engine machine	Select an existing Flow Engine machine from the Flow Engine Machines list, and click the Modify button. The Configure Flow Engine Machine dialog is displayed.
Delete a Flow Engine machine	Select an existing Flow Engine machine from the Flow Engine Machines list, and click the Delete button.
Exit the dialog	Click the Close button.
Receive additional information on using the dialog	Click the Help button.

You create or modify Flow Engine machines using the Configure Flow Engine Machine dialog. You access the Configure Flow Engine Machine dialog by clicking the New or Modify buttons on the Flow Engine Machines dialog. The Configure Flow Engine Machine dialog appears as shown in the following illustration.

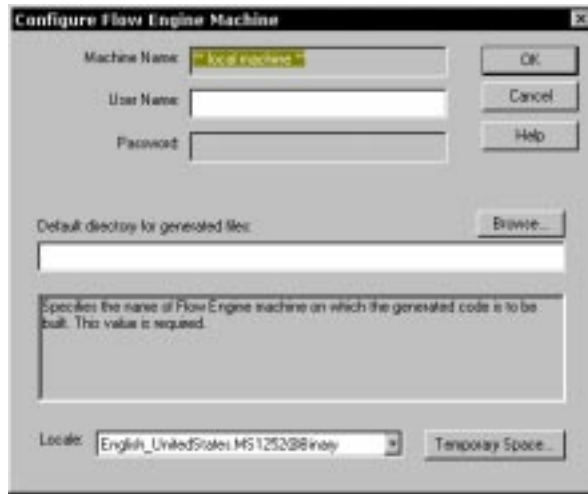


Figure 8-3
*The Configure Flow
Engine Machine
Dialog*

The following list describes the information you need to provide to the Configure Flow Engine Machine dialog to define a Flow Engine machine you can later select to generate and execute a job:

- The Machine Name field lists the TCP/IP host name of the computer where the Flow Engine component is installed and where the job will be executed. This value is required.
- The User Name field lists the user name to use when logging in to the computer named in the Machine Name field. This value is optional. If you do not specify a value, you are prompted for a user name when the job files are generated.
- The Password field lists the password associated with the user name specified in the User Name Field. This value is optional. If you do not specify a value, you are prompted for a password when the job files are generated.



Important: *The password provided is not encrypted by the Formation Architect component.*

- The Default Directory field lists the directory where the job files will be copied and built. This value is optional. If you do not specify a value, you must specify a directory when the job files are generated. If the specified directory does not exist, it will be created.
- The Locale field lists the locale assigned to the Flow Engine machine. For more information on locales, refer to [Chapter 4, “Defining the Data.”](#)
- Click the Temporary Space button to define one or more directories to use for temporary space during the processing of the job. The Temporary Space dialog is displayed. For more information on defining temporary space, refer to [“Defining Temporary Space for Job Execution” on page 8-8.](#)

Click OK to save your changes and exit the dialog.

Click Cancel to close the dialog without saving your changes.

Defining Temporary Space for Job Execution

You should define a temporary working directory for your job. This working directory is used to contain temporary intermediate files that the Flow Engine component creates during the processing of a job. In general, the more complex the job and the larger the size of the data being processed, the more temporary space you will need for a job. If no temporary space is defined, the job execution may fail.

The following sections describe:

- [Using the Temporary Space Dialog](#)
- [Default Temporary Space Locations](#)
- [Guidelines for Setting Temporary Space](#)

Using the Temporary Space Dialog

You specify the one or more directories to use to contain temporary files using the Temporary Space dialog. You access the Temporary Space dialog by clicking the Temporary Space button on the Configure Flow Engine Machine dialog. The Temporary Space dialog appears as shown in the following illustration.

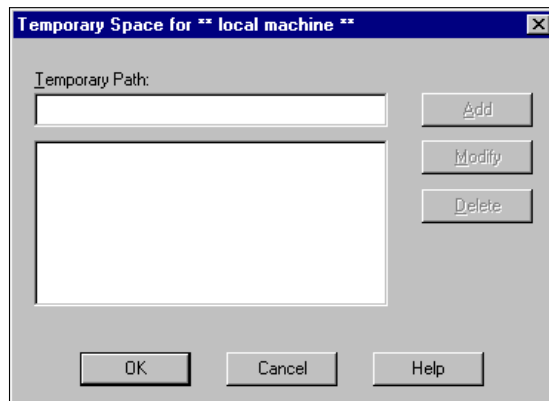


Figure 8-4
*The Temporary
Space Dialog*

The Temporary Path field lists the path you are defining or modifying. The box below it displays all temporary spaces defined for the Flow Engine machine you are currently editing. If you define more than one temporary space, the spaces are used in the order defined (starting at the top of the list) whenever the job being executed requires temporary space.

The following table lists the tasks you perform using the Temporary Space dialog.

To...	...Do the Following
Create or add a temporary space definition	Enter the temporary space directory in the Temporary Path field and click the Add button.
Modify a temporary space definition	Select an existing temporary space definition from the Temporary Path list, and click the Modify button. The definition is displayed in the Temporary Path field for editing.
Delete a temporary space definition	Select an existing temporary space definition from the Temporary Path list, and click the Delete button.
Exit the dialog	Click the OK button to save your changes and exit the dialog, or click the Cancel button to exit the dialog without saving your changes.
Receive additional information on using the dialog	Click the Help button.

Default Temporary Space Locations

If no temporary space directory is defined in the Temporary Space dialog, the Flow Engine attempts to locate temporary space in the following locations on the following operating systems.

On UNIX systems, the Flow Engine searches for the environment variables or directories in the following order:

1. If the environment variable RBF_TEMP is defined, the directory indicated by that environment variable is used.

2. If the environment variable RBF_TEMP is not defined, the /tmp directory is used.

If the /tmp directory is not available, the environment variable RBF_TEMP is not defined, and the job uses temporary space on the disk during the processing of a job, the Flow Engine component will signal an error during the execution of the job and terminate.

On Windows NT systems, the Flow Engine searches for the environment variables or directories in the following order:

1. If the environment variable RBF_TEMP is defined, the directory that it indicates is used.
2. If the environment variable TEMP is defined, the directory that it indicates is used.
3. If the environment variable TMP is defined, the directory that it indicates is used.
4. If the environment variable TMPDIR is defined, the directory the directory that it indicates is used.
5. If none of the preceding environment variables are defined, the C:\temp directory is used.

If the C:\temp directory is not available, none of the previous environment variables are defined, and the job uses temporary space on the disk during the processing of a job, the Flow Engine component signals an error during the execution of the job and terminates.

Guidelines for Setting Temporary Space

You define a temporary working directory to contain temporary intermediate files that the Flow Engine component creates during the processing of a job. When you specify a temporary space directory, you need to know how much space you need in that directory. In general, the more complex the job and the larger the size of the data being processed, the more temporary space you need for a job. If insufficient temporary space is defined, the job execution may fail.

Only certain operators require temporary space either in memory or on disk. If the operators you use in your job have the Memory Usage parameter, increasing this parameter reduces the amount of temporary space required for processing by that operator. As with most software products, you should do as much processing as possible in memory rather than on disk, because storing and retrieving information from memory is faster than storing and retrieving information from disk.

You can approximate the maximum amount of temporary space or memory your job will require by reviewing which operators are used in the job. The following table lists the operators that require temporary space or memory and lists the maximum amount of temporary space or memory each operator requires.

It is important to recognize that the maximum sizes listed in the following table are required only while the operator is processing data. Once the operator has completed the data processing, the space requirements no longer exist.

Operator	Maximum Memory or Temporary Space Requirements
Advanced Join and Join	The size of the right input data flow.
Cross Product Join	The size of both the left and right input data flows added together.
Deduplicate	The size of the input data flow.
Group By	The size of the input data flow. The more groups that you need to group by, the more the temporary space increases to a maximum of the size of the complete input data flow.
Household	The size of the input data flow.
Sort	The size of the input data flow.

Additionally, certain data flow diagram constructions can cause buffering to occur in the job, and this buffering also requires temporary space in memory or on disk. The following are the most common constructions that cause buffering:

- If an export operator has the Execution Order parameter set to a value greater than 1, the data from that operator is buffered until it can be loaded.
- If a Partition operator is followed by a Gather operator with zero or more intervening operators, the input data to the Gather is buffered.
- If a data flow is separated into two or more data flows and then recombined using the Join or Union operators, then buffering occurs at the input to the Join or Union operators. The Join operator buffers the left input data flow and the Union operator buffers two or more input data flows.

Generating and Running a Job

In This Chapter	9-3
Generating and Building a Job	9-3
Selecting the Computer and Directory on Which to Build	9-5
Generating the Source Code and Runtime Parameters Files	9-5
Building the Job	9-8
Running a Job	9-9
Setting the Flow Engine Execution Environment	9-10
Environment Variables Required to Run Jobs	9-10
Setting the Environment to Run Jobs on Windows NT Systems	9-11
Setting the Environment to Run Jobs on UNIX Systems	9-11
Running the Master Job Executable	9-12
Reviewing and Troubleshooting Job Results	9-13
Collecting and Reviewing Job Execution Information	9-15
Using the Log Utility	9-16
Using the Log Utility on UNIX Systems	9-17
Using the Log Utility on Windows NT Systems	9-19
Viewing Job Execution Information	9-21
Understanding Log Messages	9-24
Message Category Levels	9-25
Message Severity Levels	9-26
Using Log Files	9-26
Modifying the Formation Log Utility Configuration	9-27
Setting the Startup State	9-27
Specifying the Location of Log Files	9-28
Specifying the Maximum Log File Size	9-28
Setting the Log Severity Filter Level	9-29
What Happens When the Log Utility is Not Running.	9-29

Common Job Problems	9-30
Database Not Available	9-30
C++ Reserved Words Were Used as Identifiers in the Job	9-31
Invalid Flow Engine Machine Used	9-31
Invalid Field Delimiter Used in a Data Store	9-32

In This Chapter

This chapter describes how to generate, build and run a job on Windows NT and UNIX platforms. This chapter contains the following sections:

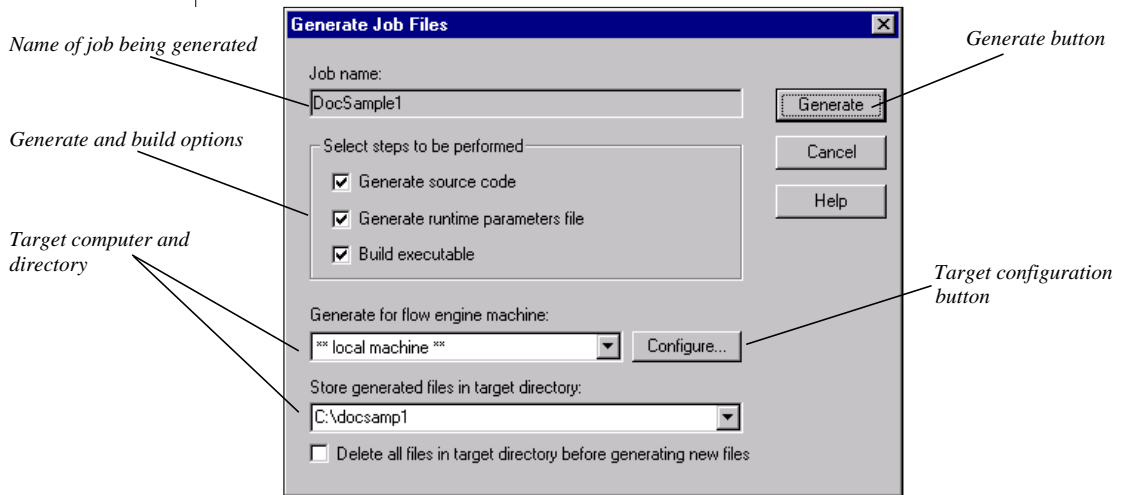
- [Generating and Building a Job](#)
- [Running a Job](#)
- [Collecting and Reviewing Job Execution Information](#)
- [Common Job Problems](#)

Generating and Building a Job

After you have completed designing a job, the next steps are to generate the source code files and build the source code files into executable files that are run with the Formation Flow Engine. You generate the source files and build the executable files by selecting Generate Job Files from the Tools menu.

The Generate Job Files dialog is displayed as shown in the following illustration.

Figure 9-1
Generate Job Files Dialog



The Generate Job Files dialog has the following key parts:

- The name of the job being generated is listed in the Job name area. This name is provided by the Formation Architect component and is the name of the current job.
- The options you can select for generating and building the job can be selected in the Select Steps to be Performed area.
- You select the computer and directory on which the job will be generated and built in the Generate and Store areas.
- You click the Generate button to actually perform the selected generation or build steps.
- You click the Configure button to add, modify or delete a computer on which you can generate and build a job. For more information on configuring a job for a new target computer, refer to [Chapter 5, “Creating Data Flow Diagrams.”](#)

The following is the process you follow in generating and building a job:

1. Specify the target computer and directory. Configure a new Flow Engine machine if needed.
2. Generate the source code and runtime parameters files.
3. Build the job executables from the source code and runtime parameter files.

The following sections describe each of these steps in more detail.

Selecting the Computer and Directory on Which to Build

The first step in generating and building a job is to select a computer on which the Formation Flow Engine is installed, and a directory on that computer. You select a computer by choosing one of the previously defined computers listed in the Generate area of the Generate Job Files dialog. You add, delete, or modify entries in this list by clicking the Configure button. For more information on configuring a job for a new target computer, refer to [Chapter 5, “Creating Data Flow Diagrams.”](#)

You select a directory on the computer by selecting it from the directories listed in the Store area of the Generate Job Files dialog, or by deleting an existing directory specification and typing in a new directory specification. The directory you select is where the generated files will be created. You should place the files for each job in a separate directory, so that the common files used by all jobs are not overwritten. If the directory you specify does not exist, it will be created.

Generating the Source Code and Runtime Parameters Files

After you have specified where the generated code should be placed, select the generation steps to perform. There are two kinds of generated files used by a job:

- Source code files
- Runtime parameter files

Source code files contain C++ source code, and runtime parameter files contain job parameter data that is used by the source code files. There are two runtime parameter files for each job, `rbf_profile.dat` and `rbf_machine.dat`. The runtime parameter files contain information about the current job operators and machine configuration, respectively. For more information on the runtime parameter files, refer to [“Examining and Modifying Runtime Parameter Files” on page 10-3](#).

You should generate source code files and runtime parameter files at the same time to ensure that these two types of generated files are synchronized. Executing a job with runtime parameter files generated from one version of a job, and with source code files generated from another version of the job could lead to unpredictable results or errors.

To generate the source code and runtime parameter files, do the following In the Select steps to be performed area of the Generate Job Files dialog:

1. Click the Generate source code and the Generate runtime parameters file boxes. Make sure that the Build Executable box is not checked, unless you want to immediately build the executables as well.
2. Click the Generate button.

When you click the Generate button, the job is verified to be valid, as if you had selected Validate from the Tools menu of the Formation Architect component. If it is not a valid job, one or more validation messages are displayed. If it is a valid job, the Build Status dialog is displayed as shown in the following illustration.

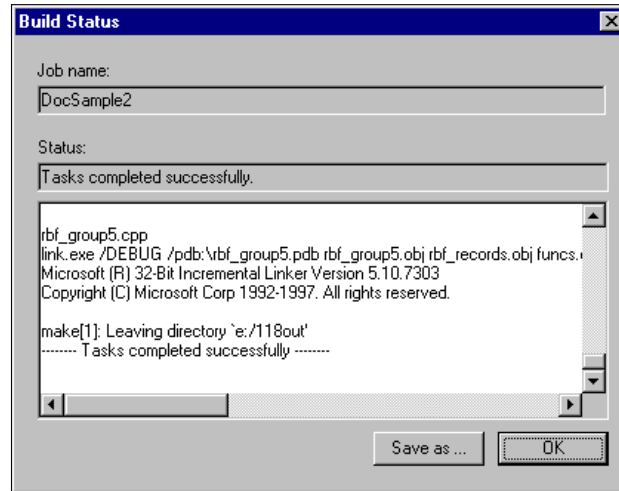


Figure 9-2
*The Build Status
Dialog*

The source code and runtime parameters files have been generated successfully when you see the following line displayed in the Build Status dialog:

```
----- Tasks completed successfully -----
```

Click the Save As button to copy the informational messages in the Build Status dialog window to an external text file. This is useful if you want to review the messages later.

Click the OK button to exit the Build Status dialog.

You can view the directory where the files were created, to verify that the files were created.



Building the Job

After you have generated the source code and runtime parameter files, you can build the job executable files. When you build the job, you compile and link the C++ files associated with the job with libraries in the Formation Flow Engine to create the executables. The build procedure uses GNU Make to control the building of the job executables. GNU Make is a utility installed as part of the Formation Flow Engine installation.

Important: *If you have another make utility installed, ensure that the appropriate make utility is used whenever you build an executable. Using a make utility other than GNU Make with the Flow Engine component could cause the automated build process to fail.*

To build the existing job source and runtime parameter files, do the following:

1. Verify that the RBF_HOME environment variable is defined on the computer on which you will build the job.

You verify that the RBF_HOME environment variable is defined to be the location of the Formation Flow Engine directory by opening a command window on the computer on which the job is to run. On Windows NT systems, you can verify the environment variables using the set command from the Command Prompt application. On UNIX systems, you can verify environment variables using the env or printenv command at the command prompt.
2. If the RBF_HOME environment variable is not defined, define it.

On Windows NT systems, define this environment variable using the Environment tab of the System application in the Control Panel.

On UNIX systems, define this environment variable interactively or by entering it into the user's shell startup script.

The shell startup script is the \$HOME/.cshrc script for C shell users and the \$HOME/.profile script for Bourne and Korn shell users. For example, if the Flow Engine is installed in the /opt/FlowEngine directory, a C shell user would add the following to their .cshrc file:

```
setenv RBF_HOME = /opt/FlowEngine
```
3. Click the Build executable box, and make sure that the other two boxes in the Select steps to be performed area in the Generate Job Files dialog are not checked.

4. Click the Generate button.

When you click the Generate button, the makefile script is run by GNU Make to compile and link the job files to create the job file executables. If the build is successful, you see the following line in the Build Status dialog:

```
----- Tasks completed successfully -----
```

Click the Save As button to copy the informational messages in the Build Status dialog window to an external text file. This is useful if you want to review the messages later.

Click the OK button to exit the Build Status dialog.

You can view the directory where the files were created, to verify that the executables were created.

Running a Job

The build process compiles and links the job files so that they are ready for execution. After you have built the job file executables, the final step is to run the job files. The Formation product supports building the job files on remote UNIX systems as well as on local Windows NT systems. You run the job executable files from the operating system prompt, outside of the context of the Formation Architect component.

To run a job, do the following:

1. Verify that the necessary environment variables are defined on the computer where the job will be run and where the Flow Engine is installed.
2. Run the master job executable.
3. Review the job logs and results to ensure that the job performed correctly.

The following sections describe each of the previous steps in detail.

Setting the Flow Engine Execution Environment

The Formation environment variables must be set correctly on the UNIX or Windows NT Flow Engine installation prior to running a job. If you are using a database with Formation, you should also verify that the environment variables for that database, if any, are also defined and have the correct values.

You verify that the necessary environment variables are set by opening a command window on the computer on which the job is to run. On Windows NT systems, you can verify the settings of the Formation environment variables using the set command from the Command Prompt application. On UNIX systems, you can verify the settings of the Formation environment variables using the env or printenv command at the command prompt.

The following sections describe:

- [Environment Variables Required to Run Jobs](#)
- [Setting the Environment to Run Jobs on Windows NT Systems](#)
- [Setting the Environment to Run Jobs on UNIX Systems](#)

Environment Variables Required to Run Jobs

The following table lists the environment variable values that must be defined to run jobs with the Formation Flow Engine.

Environment Variable	Description
LD_LIBRARY_PATH	Specifies the <code>\lib</code> subdirectory of the directory indicated by RBF_HOME.
RBF_BUILD_OPTION	Specifies an internal value. Do not modify or use.
RBF_CONFIG	Specifies the directory containing the <code>rbf.config</code> file. This file contains configuration information for the Formation product. By default, this is the same directory as where the Flow Engine software was installed.

(1 of 2)

Environment Variable	Description
RBF_HOME	Specifies the directory where the Formation Flow Engine software was installed.
RBF_HOST	Specifies an optional name used to identify the Formation Log Utility process. If this environment variable is not defined, the name is RBF_HOST. Note that this environment variable has nothing to do with the network.
RBF_ORACLE	Specifies the version of the Oracle Server interface used by the Formation product. When used with Oracle8 Server, the value of this environment variable should be 8. When used with Oracle7 Server, the value of this environment variable should be 7.

(2 of 2)

Setting the Environment to Run Jobs on Windows NT Systems

To run a job on a Windows NT system, the environment variable RBF_HOME must be defined to be the directory where the Formation Flow Engine is installed. You define this environment variable using the Environment tab of the System application in the Control Panel. All other environment variables should be defined when the Flow Engine was installed.

Setting the Environment to Run Jobs on UNIX Systems

To run a job on a UNIX system, the environment variable RBF_HOME must be defined to be the directory where the Formation Flow Engine is installed. Typically this variable is defined in the user's shell startup script. The shell startup script is the `$HOME/.cshrc` script for C shell users and the `$HOME/.profile` script for Bourne and Korn shell users. For example, if the Flow Engine is installed in the `/opt/FlowEngine` directory, a C shell user adds the following to the user's `.cshrc` file:

```
setenv RBF_HOME = /opt/FlowEngine
```

In addition, the user must define other environment variables. To define the additional environment variables, the user must use one of the following scripts that were created in the RBF_HOME directory during the installation of the Flow Engine. These scripts contain the same information but are different so they can work with different UNIX shells:

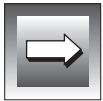
- `set_rbf_environment.csh` is used with the C shell
- `set_rbf_environment.sh` is used with the Bourne or Korn shell

To use the `set_rbf_environment.csh` script using the C shell, issue the following command, or place the command in your `.cshrc` or `.login` file:

```
source $RBF_HOME/set_rbf_environment.csh
```

To use the `set_rbf_environment.sh` script using the Bourne or Korn shell, issue the following command, or place the command in your `.profile` file:

```
.$RBF_HOME/set_rbf_environment.sh
```



Important: *The Formation Architect uses only the `set_rbf_environment.sh` file when remotely building Formation job files. Only modifications to this file will be used by the Formation Architect. Modifications to the `set_rbf_environment.csh` file will not be used by the Formation Architect.*

Running the Master Job Executable

Multiple executable files are generated as part of a job. Therefore, the Formation Flow Engine component uses a master executable file, `rbf_job`, that coordinates the running of the other executable files in the job.

The other generated executable files have names of the form `rbf_groupn`, where *n* is some whole number, such as `rbf_group4`. When you run the job by executing the master executable file you can specify a jobname parameter to the executable as follows:

```
rbf_job [jobname]
```


The `jobname` parameter is a user-defined name for that particular job run. When a job is run, that job execution is given a name so that it can be identified by the Formation software and by the user. This name is either provided by default or provided by the user as the `jobname` parameter to the job. By default, the job name is generated by appending the process identification of the running job to the name assigned to the job in the Formation Architect component.

For example, to execute a job and assign the name `run12` to that execution of the job, you would enter the following command:

```
rbf_job run12
```

This job name can also be used with the Formation Log Viewer to view the log entries for that particular job run. For example, to use the Formation Log Viewer to see the log entries from the job execution named `run12` in the active log file, you would enter the following command:

```
rbflogview -a -i run12
```

For more information on using the Formation Log Viewer, refer to [“Viewing Job Execution Information” on page 9-21](#).

When the job completes, the system command prompt is displayed.

Reviewing and Troubleshooting Job Results

After the job has been executed, you should review the job results to ensure that the job performed correctly. You can review the job results in the following ways:

- Compare the output data to the input data to see if the job processed the data as you expected.
- Review the job execution log entries using the Formation Log Utility to see if the job completed successfully. For more information on using the Formation Log Utility, refer to [“Collecting and Reviewing Job Execution Information” on page 9-15](#).

If you are not sure the job performed correctly, you can gather additional information on the job execution as follows:

- Gather intermediate processing results to review by modifying the data flow diagram and adding File Export operators at intermediate points in the data flow diagram, and then generating, building, and running the job again.
- If you are using a Red Brick Export or Red Brick Import operator, use the Message File parameter to those operators to direct messages written by the Red Brick Warehouse Table Management Utility (TMU) to a text message file for you to review. Typically, this text message file will contain more detail than the messages the TMU sends to the Formation Log Utility.
- Add your own messages to the Log Utility by using the Write Message To Log Utility function in the visual snippets you create. This function is listed in the System Functions folder.

- Enable the collection of more detailed log entries by editing the `rbf_profile.dat` file with a text editor to change the default values in that file. Once the values have been changed, rerun the job to collection additional information. The following are the variables you can change:
 - Change the value of the `Log_DebugLevel` variable from the default value 0 to 7 to enable additional informational messages about the job.
 - Change the value of the `Stat_Collection` variable to `TRUE` and the `Stat_LoggingFrequency` variable to 1 to enable additional performance messages about each operator in the job. Both of these variables must be changed for the additional performance messages to be issued.

The following fragment of a `rbf_profile.dat` file shows the `Log_DebugLevel` variable with a value of 7, the `Stat_Collection` variable with a value of `TRUE`, and the `Stat_LoggingFrequency` variable with a value of 1:

```
#Log information
[INT Log_InfoLevel 1]
[INT Log_DebugLevel 7]
[INT Log_Facility 0]
#Performance Statistics
[BOOL Stat_Collection TRUE]
[INT Stat_LoggingFrequency 1]
```

For more information on the `rbf_profile.dat` file, refer to [“Examining and Modifying Runtime Parameter Files”](#) on page 10-3.

Collecting and Reviewing Job Execution Information

You collect information about the execution of a job using the Formation Log Utility. The Formation Log Utility collects execution information and places that information in a log file during the processing of a job. This information allows you to determine whether the data is being processed correctly and helps you to diagnose error conditions. Once the log file is written, the Formation Log Viewer can be used to view the file.

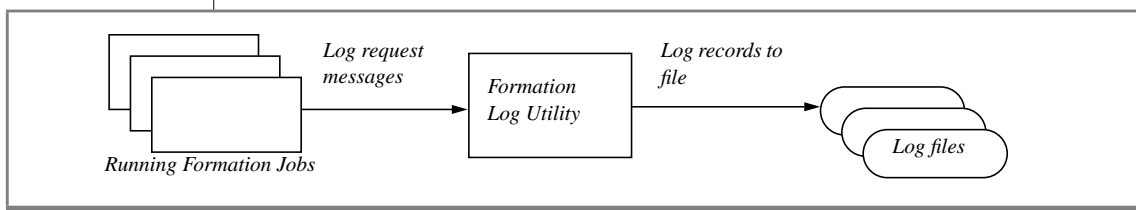
This section contains the following sections:

- [Using the Log Utility](#)
- [Viewing Job Execution Information](#)
- [Understanding Log Messages](#)
- [Using Log Files](#)
- [Modifying the Formation Log Utility Configuration](#)
- [What Happens When the Log Utility is Not Running](#)

Using the Log Utility

The Log Utility runs on Windows NT and UNIX systems, and is similar to the Red Brick Warehouse logging subsystem. On UNIX systems, event messages are collected using a logging daemon, and on Windows NT systems, event messages are collected using a logging service. On any platform, the log files are read using the Formation Log Viewer.

When the Log Utility is running, all jobs can send log request messages to the Log Utility. When a job generates and sends a log request message, it does not wait for a response. To minimize performance impact, all communication between the job and the Log Utility is one-way. For example, when a job is started, the job sends a message to the Log Utility and then continues its processing. The following figure illustrates the role of the Log Utility:



When the Log Utility receives a log request message from a job, it adds a timestamp and writes the information contained in that message to a log file. There is at most a single active log file at any given time. If the Log Utility is not running, there is no active log file. The Log Utility closes the file when the disk space limits specified have been reached, and initializes a new file.

You specify how disk space limits by using the ADMIN LOG_MAXSIZE parameter in the rbf.config file. For more information on using ADMIN LOG_MAXSIZE, refer to [“Modifying the Formation Log Utility Configuration” on page 9-27](#).

For more information on log files, refer to [“Using Log Files” on page 9-26](#). For more information on Log Utility configuration parameters, refer to [“Modifying the Formation Log Utility Configuration” on page 9-27](#).

The following sections describe using the Log Utility on UNIX and Windows NT systems in detail.

Using the Log Utility on UNIX Systems

On UNIX systems, the Log Utility is implemented as a log daemon process (rbflogd) that handles log request messages issued by Formation jobs when various events occur. The Formation log daemon is controlled by executing several scripts that are installed in the /bin subdirectory of the directory in which you installed the Formation Flow Engine.

The following table lists the tasks you can perform to control the Formation Log Utility daemon.

To...	Do the Following...
Start a Formation Log Utility daemon	<p>Run the <code>rbf.start</code> script to start a Formation Log Utility daemon. The <code>rbf.start</code> script has the following syntax:</p> <pre>rbf.start config-dir host-name</pre> <p>When running this script, you must provide both parameters. These parameters are described in the following table. If the <code>RBF_CONFIG</code> and <code>RBF_HOST</code> environment variables are defined, the script can be specified as the following:</p> <pre>rbf.start \$RBF_CONFIG \$RBF_HOST</pre>
Stop a Formation Log Utility daemon	<p>Run the <code>rbf.stop</code> script to stop a Formation Log Utility daemon. The <code>rbf.stop</code> script has the following syntax:</p> <pre>rbf.stop host-name</pre> <p>When running this script, you must provide the <i>host-name</i> parameter. This parameters is described in the following table. If the <code>RBF_HOST</code> environment variable is defined, the script can be specified as the following:</p> <pre>rbf.stop \$RBF_HOST</pre>
Show one or all Formation Log Utility daemons	<p>Run the <code>rbf.show</code> script to show all or only the named Formation Log Utility daemon. The <i>rbf.show</i> script has the following syntax:</p> <pre>rbf.show [-n host-name]</pre> <p>When running this script, if you do not specify a parameter, all running Formation Log Utility daemons are listed. If you provide the <code>-n</code> option with the <i>host-name</i> parameter, only the named Formation Log Utility daemon is listed. If the <code>RBF_HOST</code> environment variable is defined, the script can be specified as the following to display a single named daemon:</p> <pre>rbf.show -n \$RBF_HOST</pre> <p>To display all running Formation Log Utility daemons, specify the script as follows:</p> <pre>rbf.show</pre>

The following table describes the options and parameters used in the previous syntax statements for the `rbf.start`, `rbf.stop`, and `rbf.show` scripts:

Option or Parameter	Description
<code>config-dir</code>	Specifies the directory where the <code>rbf.config</code> file is located. This should be the same directory indicated by the environment variable <code>RBF_CONFIG</code> .
<code>host-name</code>	Specifies the name given to the log daemon. This should be the name specified by the environment variable <code>RBF_HOST</code> .
<code>-n host-name</code>	Specifies the name given to the log daemon with the <code>rbf.show</code> script.

Using the Log Utility on Windows NT Systems

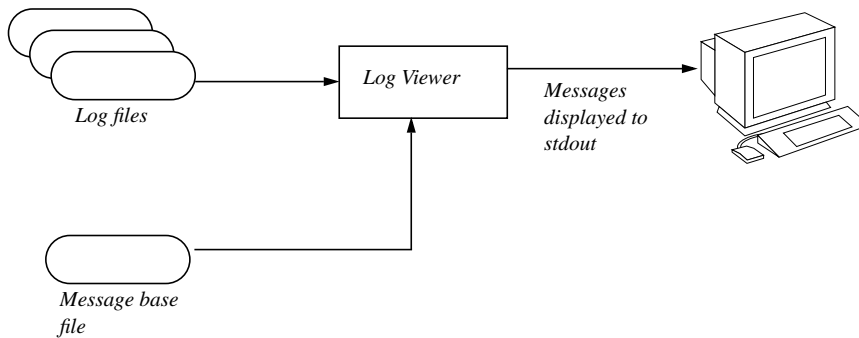
On Windows NT systems, the Log Utility is implemented as a log service that handles log request messages issued by Formation when various events occur. The Formation Log service works just like any other Windows NT service. The Formation Log service is started automatically when you start the Formation product.

You control the Formation Log service as you would any Windows NT service, by using the Services applet in the Windows NT Control Panel. The account from which you manipulate the Formation Log Utility service should be a member of the Administrator group. The following table lists the tasks you can perform to control the Formation Log Utility service:

To...	Do the Following...
Start the Formation Log Utility service	Select the Informix Formation Log Utility service from the list of services in the Services applet and click the Start button.
Stop the Formation Log Utility Service	Select the Informix Formation Log Utility service from the list of services in the Services applet and click the Stop button.
Show the status of the Formation Log Utility service	Select the Informix Formation Log Utility service from the list of services in the Services applet and view its status.

Viewing Job Execution Information

You can use the Formation Log Viewer to view the contents of the log. The log service or daemon writes the parameter values contained in a log message to the log file. The log service or daemon does not write full message text to the log file. This text is stored in the form of message templates in a separate message base file. When you view the event messages, the Log Viewer combines the appropriate message template with the parameter values stored in the log file to give you readable output. The resulting message is displayed to stdout or written to a file. For a discussion of the message parameters, refer to [“Understanding Log Messages”](#) on page 9-24.



The Log Viewer executable is named `rbflogview`. Any user who has read permission for the log files can view event messages using the Log Viewer. The Log Viewer has the same syntax on UNIX and Windows NT systems, as follows:

```
rbflogview [-a] [-t] [-e] [-f] [-p pid]
           [-i jobname [[-i jobname] ...]]
           [-c [e][o][j][p][x]]
           [-s [a][r][u]]
           [logfile [[logfile] ...]]
```

The following are the descriptions of the options and parameters to the `rbflogview` command described in the previous syntax statement:

Option	Description
<code>-a</code>	Specifies the active log file. If you use this option you cannot specify the <i>logfile</i> parameter.
<code>-t</code>	Specifies terse output with shortened headers.
<code>-e</code>	Specifies continuous display of the active log file. As new records are written to the active file they are displayed to <i>stdout</i> . This is similar to the UNIX system <i>tail</i> command.
<code>-f</code>	Specifies continuous display of active log file. All records currently in the active file are first displayed, followed by any new records as they are written to file. This is similar to the UNIX system <i>tail</i> command.
<code>-p pid</code>	Displays only those log records that originate from the job with the specified process identification.
<code>-i jobname</code>	<p>Displays only those log records that originate from the one or more jobs that were run with the specified <i>jobname</i>.</p> <p>When a job is run, that job run is given a name so that it can be identified by the Formation software. This name is either provided by default or provided by the user as a parameter to the job. By default, the job name is generated by appending the process identification of the running job to the name of the job in the Formation Architect component.</p> <p>For example, to view the log records from the job run named <code>run12</code> in the active log file, you would enter the following command:</p> <pre>rbflogview -a -i run12</pre>

(1 of 2)

Option	Description
-c	Limits display to the specified categories, one or more of the following: e (error), j (job), o (operational), p (operator) or x (external). For more information on event categories, refer to “Message Category Levels” on page 9-25 .
-s	Limits display to the specified severities, one or more of the following: u (urgent), a (alert), or r (routine). For more information on event severities, refer to “Message Severity Levels” on page 9-26 .
logfile	Specifies a particular log file to read from. Multiple files can be specified and will be read in the order that they were saved by the log service or daemon. For more information on log files, refer to “Using Log Files” on page 9-26 .

(2 of 2)

You must specify either one or more closed log files or the active log file as Log Viewer input; otherwise the Log Viewer reads from stdin, which is typically the window in which you are running the Log Viewer. The -a option causes the Log Viewer to read from the active log file. You cannot specify both a closed log file and the -a option.

The Log Viewer writes to stdout, which is typically the window in which you are running the Log Viewer. In the following example of using the Log Viewer, the following command writes all log records in the closed log file named *rbflog.smoky.980421.101053* to the window in which you are running the Log Viewer.

```
rbflogview rbflog.smoky.980421.101053
```

The following is the output from the previous command:

```
Sep 05 16:12:35 run12[225] OPE2503A: [run12|main|Information]
Informix Formation (TM) BL999.0 Sep 4 1998
Node name: XLSIOR
Process ID: 241

Sep 05 16:12:35 run12[225] OPE2503A: [run12|main|Information]
Environment Variables:
RBF_HOME=e:/FORM/server/
RBF_CONFIG=e:/FORM/server/
```

```

RBF_JOB= ./

Sep 05 16:12:35 run12[221] OPE2503A:
[run12|rbf_group0|Information]
Informix Formation (TM) BL999.0 Sep 4 1998
Node name: XLSIOR
:
```

Often, when you expect a large amount of output from the Log Viewer, you will want to redirect the output of the Log Viewer. You do this by piping the output to the more command so that the output is displayed a page at a time, or you can redirect it to a text file.

The following example shows how you would use the Log Viewer with the more command.

```
rbflogview rbflog.smoky.980421.101053 | more
```

The following example shows how you would redirect the output of the Log Viewer to a text file named mylog.txt.

```
rbflogview rbflog.smoky.980421.101053 > mylog.txt
```

Understanding Log Messages

When viewed using the Log Viewer, all messages have the following form:

```
CCC####S: Message Text
```

The first three characters (CCC) indicate the message category, the next four numeric characters (####) represent the message number, and the last character (S) indicates the message severity. The message text follows the message code. The following is an example of a message:

```

Aug 05 16:12:37 run12[228] OPE2503A:
[run12|rbf_group5|Information]
Process 'rbf_group5' startup
```

In the previous example, the message code is OPE2503A. This message code identifies the following about the message:

- The message has a category of operational, based on the OPE message category code.
- The message has a number of 2503.
- The message has a severity of alert, based on the A message severity code.

- The message text is “*Process 'rbf_group5' startup.*”

The following sections discuss message category and severity levels in detail.

Message Category Levels

Event messages are assigned one of the following categories:

Message Category and Prefix	Identifies...
ERROR (ERR)	Error, warning, or exception messages about user actions or changes in the Formation system environment. The default minimum severity level is ROUTINE.
EXTERNAL (EXT)	Informational messages about external applications used by the Formation product. The default minimum severity level is ROUTINE.
JOB (JOB)	Informational messages about a Formation job. The default minimum severity level job is ROUTINE.
OPERATIONAL (OPE)	Informational messages about the administration of the Formation product, such as component startup or shutdown. These changes are usually done by warehouse administrators or other users in similar roles. The default minimum severity level is ALERT.
OPERATOR (OPR)	Informational messages about an operator in the Formation product. The default minimum severity level is ROUTINE.

You can specify a minimum severity level for logged records in each log category. For example, you could specify that the minimum severity for ERROR events is ALERT. In this case, only those ERROR events that have ALERT or URGENT severity are logged. To specify the minimum severity level for an event category, you set the appropriate configuration parameter directly. For more information on setting the log severity level, refer to [“Setting the Log Severity Filter Level” on page 9-29](#).

Message Severity Levels

Each event message has an event severity level, which is indicated by the last character of the message code in the log file. The following is an ordered list of the severity levels, with the topmost list item being the lowest severity, and the bottommost list item being the highest severity:

- ROUTINE, indicated by an R character in the message code.
- ALERT, indicated by an A character in the message code.
- URGENT, indicated by a U character in the message code.

Using Log Files

The log service or daemon writes log records to the active log file. When this file exceeds the size specified by the ADMIN LOG_MAXSIZE configuration parameter in the rbf.config file (if that parameter is entered in that file), the log service or daemon closes the file and creates a new active file. The log service or daemon also closes the active file when you stop the log service or daemon and creates a new active file when the log service or daemon starts. In this manner, a sequence of log files accumulates over time. The contents of the files in this sequence can be concatenated before processing because none of the files contain any header or trailer information.

The active log file and the saved log files have the following naming conventions:

```
rbflog.<host_name>.active  
rbflog.<host_name>.<datetime_stamp>
```

The *<host_name>* variable indicates the name specified as the value to the RBF_HOST environment variable. The *<datetime_stamp>* suffix on saved file names indicates the date and time at which the log file was closed. For example, the following would be valid names for logs of jobs on the computer, MYCOMP:

```
rbflog.MYCOMP.active  
rbflog.MYCOMP.19980421.184749
```

The default location for these files is the logs subdirectory of the directory indicated by the RBF_CONFIG environment variable.



Old log files are not removed automatically. The warehouse administrator must provide a script to periodically remove these files or remove them manually.

Important: *If you do not remove old log files, these files will accumulate over time and potentially consume all the free disk space. In addition, if you do not specify a value for ADMIN LOG_MAXSIZE in the rbf.config file, the log service or daemon will write to a single log file that grows until limited by available disk space. For more information on using ADMIN LOG_MAXSIZE, refer to “[Modifying the Formation Log Utility Configuration](#)” on page 9-27.*

Modifying the Formation Log Utility Configuration

You can modify the Log Utility configuration by setting various configuration parameters in the rbf.config file. This file is located in the directory specified as the value to the RBF_CONFIG environment variable. The following sections describe the parameters you can set.

Setting the Startup State

The ADMIN LOGGING parameter determines the startup actions of the logging service or daemon. The syntax for setting this parameter follows:

▶▶ ADMIN LOGGING ☐ ON ☐ OFF ▶▶

If the ADMIN LOGGING parameter is set to ON, the log service or daemon creates an initial log file when it initializes and starts logging events. If this parameter is set to OFF, the log service or daemon does not perform any operations.

Specifying the Location of Log Files

The ADMIN LOG_DIRECTORY parameter specifies the directory in which the log service or daemon creates the log files. The syntax for setting this parameter follows:

▶▶ ADMIN LOG_DIRECTORY — *pathname* ▶▶

The pathname can be an absolute pathname or a relative pathname. Relative pathnames are interpreted as relative to the directory specified by the RBF_CONFIG environment variable. If you do not set the ADMIN LOG_DIRECTORY parameter, the default logging directory is the logs subdirectory of the directory indicated by the RBF_CONFIG environment variable.

Specifying the Maximum Log File Size

The ADMIN LOG_MAXSIZE parameter specifies the maximum log file size. The syntax for setting this parameter follows:

▶▶ ADMIN LOG_MAXSIZE —

<i>integer</i>
<i>integerK</i>
<i>integerM</i>

 ▶▶

When a log file exceeds the size specified by this parameter, the log service or daemon closes the file and creates a new active file in the same directory. The units of the integer value are interpreted as follows:

- Bytes if neither K nor M is specified
- Kilobytes (1,024 bytes) if K is specified
- Megabytes (1,048,576 bytes) if M is specified

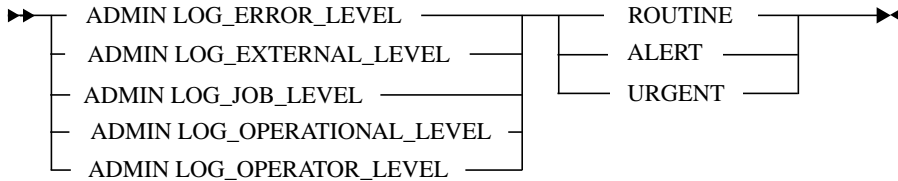
If you specify K or M, this suffix must immediately follow the numeric value with no spaces. The minimum value for this parameter is 10K (10,240) bytes.

Warning: If you do not set this configuration parameter, or if you set this parameter to zero or a negative number, no maximum size is imposed on log files. In this case, log files can continue to grow, limited only by available disk space in the log directory.



Setting the Log Severity Filter Level

There are separate configuration parameters for setting the log severity filter level, one for each message category. The log severity filter level represents the minimum severity an event within a given category must have in order to be logged. The syntax for these parameters follows:



What Happens When the Log Utility is Not Running

If the Formation Log Utility is not running, messages issued by jobs are directed to one or more files rather than to the Log Utility. The Formation Log Utility does not run when the RBF_CONFIG and RBF_HOST environment variables are not defined correctly, or if the Formation log daemon on UNIX systems or Formation log service on Windows NT systems was not started.

The message files are placed in the directory in which the job is executed. The name and number of log files generated depends on the operating system on which the job is run.

On Windows NT systems, multiple log files are created with a name in the following format:

`rbf.jobname.groupname.computername.processid.log`

On UNIX systems, a single log file is created with a name in the following format:

`rbf.jobname.computername.processid.log`

The parts of the log file name are as follows:

Log File Name Component	Description
jobname	Indicates the name given to the job when it was run. This name is either user-defined or generated by the Flow Engine component.
groupname	Indicates the name of a processing group within the job on a Windows NT system. The processing group name will either be <i>main</i> , indicating the controlling process, or will be named <i>rbf_groupn</i> , where <i>n</i> indicates a group number.
computername	Specifies the name of the computer on which the job is run.
processid	Indicates the process identification of the running job on UNIX systems or the processing group of the running job on Windows NT systems.

The following is an example log file name for a job named MyJob2 that ran on a Windows NT system:

```
rbf.MyJob2.rbf_group13.MYCOMP.34515.log
```

Common Job Problems

Sometimes a job does not work as expected, or work at all. The following are some of the more common problems associated with a job failing to perform as expected.

Database Not Available

When a job cannot read from or write to a database, it may be that the database is not installed, not configured, or not running in the location you have specified.

In particular, when using the Oracle Server database with a UNIX Flow Engine, ensure that the UNIX Flow Engine has been configured as described in the [Informix Formation Installation and Configuration Guide](#).

C++ Reserved Words Were Used as Identifiers in the Job

When C++ reserved words are used as identifiers in Informix Formation, such as in a parameter or in a snippet, unpredictable results or errors may occur when the job is generated, compiled, linked, or executed. This is because these identifiers may be misinterpreted when the C++ code generated by Informix Formation is compiled, linked, or executed.

Do not use C++ reserved words as such identifiers. The following is a list of C++ reserved words that should typically not be used (the actual list will vary by compiler):

asm1	dynamic_cast	namespace	this
auto	else	new	throw
bad_cast	enum	operator	true
bad_typeid	except	private	try
bool	explicit	protected	type_info
break	extern	public	typedef
case	false	register	typeid
catch	finally	reinterpret_cast	typename
char	float	return	union
class	for	short	unsigned
const	friend	signed	using
const_cast	goto	sizeof	virtual
continue	if	static	void
default	inline	static_cast	volatile
delete	int	struct	while
do	long	switch	xalloc
double	mutable	template	

For a complete list of the reserved words to not use, see the documentation for the C++ compiler you are using with the Flow Engine.

Invalid Flow Engine Machine Used

Invalid Flow Engine machines can cause errors in a job, although these errors do not typically cause the job to fail.

On UNIX systems, invalid Flow Engine machines can cause errors to be written to the Formation Log Utility when a job is run. On Windows NT systems, invalid Flow Engine machines can also cause the operating system to query the user for unknown computers when a job is run. For example, if you import a workspace containing a Flow Engine machine for a computer that is not on your network, when you run that job on Windows NT you may receive an operating system warning that the computer is not available.

When importing a workspace, any Flow Engine machine definitions that were part of that workspace are also imported. These Flow Engine machines are added to the list of available Flow engine machines in the Architect tool.

You should review this list after importing a workspace to ensure that the new definitions are valid for your environment, and remove those definitions that are not valid. On UNIX systems, invalid Flow Engine machines can cause errors to be written to the Formation Log Utility when a job is run. On Windows NT systems, invalid Flow Engine machines can cause the operating system to query the user for unknown computers when a job is run.

Invalid Field Delimiter Used in a Data Store

If a field is delimited in a data store with an invalid character, such as the double quote character ("), the job may fail during compilation.

Replace the invalid delimiter with a valid delimiter.

Advanced Job Topics

In This Chapter	10-3
Examining and Modifying Runtime Parameter Files	10-3
Examining rbf_machine.dat	10-3
Examining and Modifying rbf_profile.dat	10-4
Transferring Data Between Executing Jobs.	10-6
Using External Libraries in a Job	10-7
Creating an External Function.	10-7
Call the External Function from a Visual Snippet	10-9
Specify Build Options for the External Function	10-9

In This Chapter

This chapter describes the following more advanced job creation and usage topics:

- [Examining and Modifying Runtime Parameter Files](#)
- [Transferring Data Between Executing Jobs](#)
- [Using External Libraries in a Job](#)

Examining and Modifying Runtime Parameter Files

You generate the runtime parameter files using the Generate Job Files dialog. For more information on generating these files, [Chapter 9, “Generating and Running a Job.”](#) There are two runtime parameter files for each job, `rbf_machine.dat` and `rbf_profile.dat`. The runtime parameter files contain information about the current job operators and machine configuration, respectively. The following sections describe each of these files in more detail.

Examining `rbf_machine.dat`

The machine configuration runtime parameter file, `rbf_machine.dat`, contains information about the current job computer configuration. This file is read as part of the running of the executable job files.

Any changes to the computer configuration runtime parameter file, `rbf_machine.dat`, should be done using the Formation Architect component, and then you should regenerate all job files. For more information on changing the machine configuration, refer to [Chapter 5, “Creating Data Flow Diagrams.”](#)

The following is the contents of a simple rbf_machine.dat file generated from the DocSample1 job:

```
# System configuration: rbf_machine.dat
#
# Generated by: Informix Formation (TM) Version 1.4.1
# Generated at: Fri Apr 24 12:58:16 1999
#
# Copyright (c) 1998 Informix Software, Inc., Menlo Park, CA, USA.
# All rights reserved. Confidential and proprietary.
#
# Usage:
# (node <hostname> (path "<directory>"))

# Physical nodes
(node XLSIOR)
```

Examining and Modifying rbf_profile.dat

The operator runtime parameter file, rbf_profile.dat, contains information about the current job logging parameters and operator parameters. This file is read as part of the running of the executable job files. The rbf_profile.dat file has the following parts:

- An initial comment area that lists the name of the job and gives the time that the rbf_profile.dat file was generated.
- The log information area that lists what kinds of informational messages are being logged. The following are the three variables set in this area:

Variable	Value
Log_InfoLevel	Reserved for internal use, do not modify.
Log_Debug_level	Sets how detailed the status messages should be. A value of 0 indicates that only the most important status messages should be sent to the Formation Log Utility. A value of 7 indicates that all status messages should be sent to the Formation Log Utility.
Log_Facility	Reserved for internal use, do not modify.

- The parameter information for each operator in the job. The parameter information for each operator is preceded by a comment line with the name of that operator. For more information on what each parameter indicates, refer to [Chapter 6, “Selecting and Using Operators,”](#) or see the Architect online help.

You should not modify the `rbf_profile.dat` file unless you want to increase the detail on the log messages issued by the running job. All other modifications to the job parameters and parameter values, should be done in the Architect component, and then the files should be generated again.

You can enable the collection of more detailed log entries by editing the `rbf_profile.dat` file using a text editor and changing the value of the `Log_DebugLevel` variable from 0 to 7, and then running the job again.

The default value of the `Log_DebugLevel` variable is 0. The following fragment of a `rbf_profile.dat` file shows the `Log_DebugLevel` variable with a modified value of 7.

```
#Log information
[INT Log_InfoLevel 1]
[INT Log_DebugLevel 7]
[INT Log_Facility 0]
```

The following is a portion of a `rbf_profile.dat` file from the job `DocSample1`:

```
# rbf_profile.dat
# Job configuration: DocSample1
# Generated by: Informix Formation (TM) Version 1.4.1
# Generated at: Fri Aug 24 12:58:16 1999
# Copyright (c) 1998 Informix Software, Inc., Menlo Park, CA, USA.
# All rights reserved. Confidential and proprietary.
# Log information
[INT Log_InfoLevel 1]
[INT Log_DebugLevel 0]
[INT Log_Facility 0]
# Group number: 0
[BOOL rbf_group0_Debug false]
# File Import Operator: 'NewShop_POS'
[STRING NewShop_POS_FileList
 ("F:\Program Files\Informix\Formation\Samples\NewShopPOS.txt")]
[INT NewShop_POS_StartRecord 1]
[INT NewShop_POS_SkipRecords 0]
[BOOL NewShop_POS_Result_Buffer true]
# Group number: 1
[BOOL rbf_group1_Debug false]
[BOOL rbf_group1_Buffer true]
# Red Brick Export Operator: 'Red_Brick_Export_1'
[STRING Red_Brick_Export_1_Database "aroma"]
[STRING Red_Brick_Export_1_TableName "sales"]
[STRING Red_Brick_Export_1_UserName "system"]
[STRING Red_Brick_Export_1_Password "manager"]
[BOOL Red_Brick_Export_1_ParallelTMU false]
```

```
[INT Red_Brick_Export_1_LogInterval 10000]
[STRING Red_Brick_Export_1_LoadMode "Modify"]
[STRING Red_Brick_Export_1_ASCTIDiscardFile ""]
[INT Red_Brick_Export_1_MaxDiscards 0]
[STRING Red_Brick_Export_1_RefIntDiscardFile ""]
[STRING Red_Brick_Export_1_AutoRowGen "Default"]
[STRING Red_Brick_Export_1_Optimize "Default"]
[STRING Red_Brick_Export_1_OptimizeDiscardFile ""]
[STRING Red_Brick_Export_1_Segment ""]
[STRING Red_Brick_Export_1_WorkingSpace ""]
[STRING Red_Brick_Export_1_Comment ""]
[STRING Red_Brick_Export_1_CommandFile ""]
[STRING Red_Brick_Export_1_MessageFile ""]
[INT Red_Brick_Export_1_LoadPhase 1]
[BOOL Red_Brick_Export_1_Data_Buffer true]
:
```

Transferring Data Between Executing Jobs

You can transfer data between jobs by using named pipes as follows:

1. A job uses a File Export operator to export data to a named pipe.
For example, job FirstJob contains a File Export operator that specifies the following named pipe on Windows NT as the File List parameter to the File Export operator:

```
pipe:///d:\temp\mypipe.dat
```

2. Another job uses a File Import operator to import data from the named pipe already created by a previous job.

For example, job SecondJob contains a File Import operator that is used to read from the named pipe created by job FirstJob, mypipe.dat. This is done by specifying the same pipe name as the File List parameter to the File Import operator in job SecondJob:

```
pipe:///d:\temp\mypipe.dat
```



Important: The named pipe is created by the File Export operator. If the named pipe is not created before a File Import operator attempts to read from it, the job containing the File Import operator will fail.

For more information on what named pipe specifications are valid for the File Export and File Import operators, refer to [Chapter 6, “Selecting and Using Operators,”](#) or to the operator descriptions in the Architect online help.

Using External Libraries in a Job

You use external libraries to extend the capabilities of the Formation product. Support for external libraries allows you to access industry-standard warehousing tool application programming interfaces (API) and libraries, as well as your own C and C++ routines from within a Formation job. However, using external libraries requires you to be familiar with the GNU Make utility and with compiling and linking C and C++ programs. You use external libraries by defining them as external functions within the Formation product.

The following is the process for defining external functions for use by Formation jobs:

1. Create external functions using the Formation Architect component.
2. Use the external functions you created in the appropriate visual snippet in the appropriate operator.
3. Create the file `rbf_config.mk` to specify the library files, paths, or compiler flags needed to compile and link the external functions.

The following sections describe these tasks in detail.

Creating an External Function

You create an external function by performing the following steps:

1. Select New External Function from the File menu in the Formation Architect component.
2. In the dialog that is displayed, provide a name for the external function and click the OK button. The name you provide is how the function will be displayed in the Formation product and does not have to be the actual routine name.

The external function is displayed in the Custom folder in the Toolbox tab of the Navigator. A new dialog for further specifying the external function is displayed. If you selected another folder in which to place the external function, the external function is placed in that folder.

3. In the dialog that is displayed, do the following to define the external function:
 - Select C Data Types from the Types list to indicate that the function uses C or C++ datatypes.
 - Provide the name of the external function as it is declared in the library in the Routine Name box.
 - Provide the name of the include file which contains the prototype of the external function in the Includes box.
 - Modify the default external function variable name, void, so you can enter the return type of the external function. Select the default variable name, void, and click the Modify button. The External Function Parameter dialog is displayed.
4. In the External Function Parameter dialog, define the return type of the external function by doing the following:
 - Enter a variable name for the return type of the external function in the Name box.
 - Enter the datatype and passing mechanism for the variable in the C Data Type box.
 - Indicate that the return type has an output passing mode by selecting the Output box.
 - Optionally, provide a description of the return type in the Description box.

Click the OK button when you have defined the return type.

5. In the dialog that is displayed, define additional arguments by clicking the Insert After button and then doing the following for each argument:
 - Enter a variable name for the argument in the Name box.
 - Enter the datatype and passing mechanism for the argument in the C Data Type box.
 - Enter the size of the argument, if necessary.
 - Indicate the passing mode of the argument by selecting the Output box, the Input box, or both.
 - Optionally, enter a default value for the argument.
 - Optionally, provide a description of the argument in the Description box.
 - Click the OK button after you have defined the argument.

Call the External Function from a Visual Snippet

After defining the external function, you call it from a visual snippet inside of an operator. You add an external function to a visual snippet in the same way you add a standard function to a visual snippet.

Select the operator and visual snippet from which to call the external function. After selecting the appropriate visual snippet, add the external function to that snippet.

Specify Build Options for the External Function

You need to specify library files, paths, or compiler flags to compile and link the external function into your Formation executables. You specify this information in the `rbf_config.mk` file.

To specify this information, create a file named `rbf_config.mk` using a text editor and place it in the directory indicated by the `RBF_CONFIG` environment variable. This file contains the GNU Make variables needed by the external functions you are adding to the job. This file is included by the generated job make file, `rbf_job.mk`.

Warning: Do not directly edit the `rbf_job.mk` file. Doing so may cause the job to not build or to build incorrectly.



The supported variables are as follows:

GNU Make Variable	Description
EXTERNAL_LIB	Specifies the library files you would include using the #library preprocessor directive.
EXTERNAL_INC	Specifies the include file paths you would include using the #include preprocessor directive.
EXTERNAL_CFLAGS	Specifies any compiler flags you would include using the #optional preprocessor directive.

The following is a sample rbf_config.mk file for use on the Windows NT platform:

```
# External Library Make Include File for Windows NT
#####
EXTERNAL_CFLAGS :=
EXTERNAL_LIB      := e:/codegen/extlib1/debug/extlib1.lib \
                    e:/codegen/extlib2/debug/extlib2.lib
EXTERNAL_INC      := -Ie:/codegen/extlib1 -Ie:/codegen/extlib2
```

The previous example uses two libraries and does not specify any compiler flags:

- The EXTERNAL_CFLAGS variable defines no compiler flags. This variable could also be removed from this file since it is not used.
- The EXTERNAL_INC variable defines two include paths required for compiling the C functions used by the external function. These paths are e:/codegen/extlib1 and e:/codegen.extlib2.
- The EXTERNAL_LIB variable defines path references to each library. These paths are e:/codegen/extlib1/debug/extlib1.lib and e:/codegen/extlib2/debug/extlib2.lib.

Note that because GNU Make is used, the paths specified to the GNU Make variables must use the slash (/) character to delimit directories on both Windows NT and UNIX platforms, this is not the standard method of delimiting directories on the Windows NT platform.

DocSample2: An Extended Example of a Data Flow Diagram

This appendix contains a sample data flow diagram that builds on the DocSample1 job explained earlier. For more information on the DocSample1 job, refer to [Chapter 2, “DocSample1: A Quick Start Guide.”](#)

The job is explained in the following sections:

- [Analyzing the DocSample2 Job](#)
- [Defining the Data Stores](#)
- [Setting Up Import and Export Operators](#)
- [Transforming the Data](#)
- [Exporting the Data](#)
- [Running the Job](#)

This sample job uses two tables from the Aroma database, which is part of the Red Brick Warehouse product. Successfully executing this sample job requires access to the Red Brick Aroma database on a supported platform.

For detailed information on the operators mentioned in this chapter, refer to [Chapter 6, “Selecting and Using Operators,”](#) or the Architect online help.

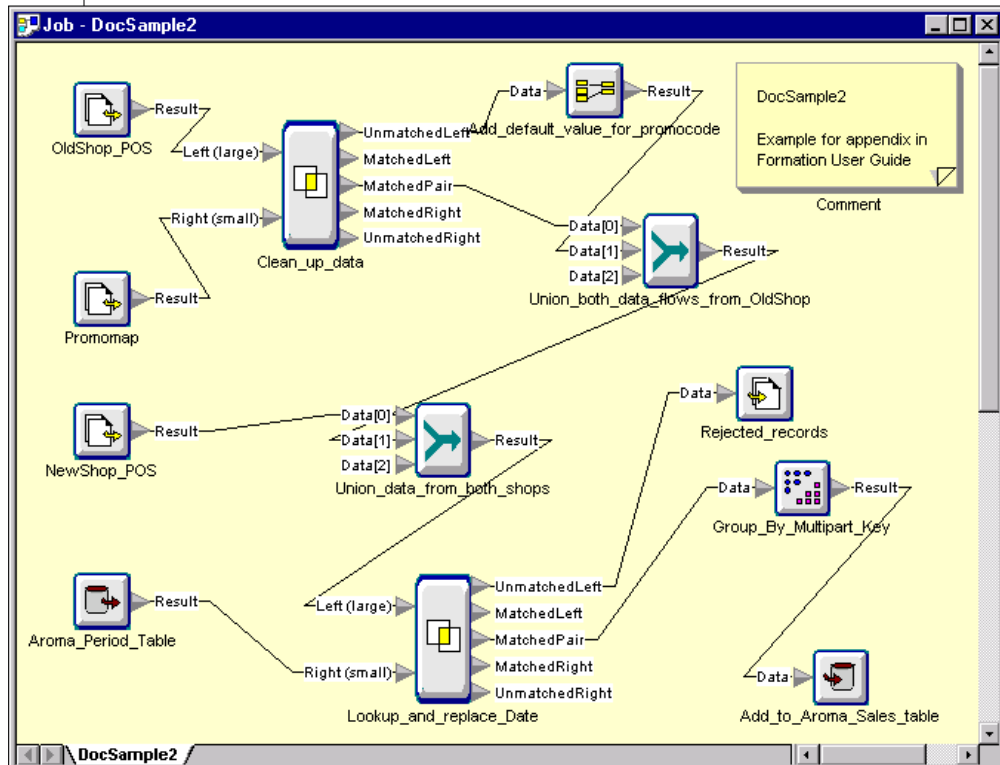
Analyzing the DocSample2 Job

The DocSample2 job is a response to a challenge many retail store chains face: the data generated by different stores does not contain the same data or use the same format and so each store's data must be transformed before it is added to the existing data warehouse.

The NewShop data is relatively clean because the only cleanup work is replacing the date field with a period key. However, the OldShop data is more problematic since it doesn't contain a code to identify the shop, uses an old set of promotion codes, and also needs the date replaced by a period key. Once the data from each shop has been modified as necessary, the two sets of data can be combined into one, and the data grouped by key as the final step before appending it to the existing Aroma Sales table.

The following illustration shows the data flow diagram for DocSample2. The job takes text files from two retail shops, transforms the data as needed, combines the data into one set, groups the data to match the Aroma Sales table, and then appends the data to the Sales table.

Figure A-1
DocSample2 Data Flow Diagram



Defining the Data Stores

The DocSample2 job requires the following data stores:

Data Store Name	Description	Type
OldShop	Data from the old shop	Text file
NewShop	Data from the new shop	Text file
PromoMap	Promotion code used by old store and the equivalent corporate promotion code	Text file
Period	Period table from Aroma database	Database table
Sales	Sales table from Aroma database	Database table

The OldShop Data Store

This shop generates records that are not quite up to corporate standards. In addition, the key that identifies the store is not specified in each record.

Field	Description	Data Type
TransactionNumber	Unique identifier for each transaction	Integer
ClassKey	First part of two-part key that identifies each product	Integer
ProductKey	Second part of two-part key that identifies each product	Integer
Date	Date in the format YYYY-MM-DD	Date
Quantity	Number of items sold	Integer
Amount	Dollar amount of sale	Decimal, with precision of two
PromoCode	Store-specific promotion code	Integer

The file that stores the OldShop data is located in the Samples directory under the main Formation directory (the default is: C:\Program Files\Informix\Formation\Samples\oldshoppoos.txt).

The NewShop Data Store

This store is new and the data is formatted to corporate standards, although the fields are not in the required order.

Field	Description	Data Type
Date	Corporate standard format (YYYY-MM-DD)	Date
ClassKey	First part of two-part key that identifies each product	Integer
ProductKey	Second part of two-part key that identifies each product	Integer
StoreKey	Identifies this store; matches corporate standard	Integer
PromoKey	Corporate standard	Integer
Quantity	Number of items sold	Integer
Amount	Dollar amount of sale	Decimal, with precision of 2

The file that stores the NewShop data is located in the Samples directory under the main Formation directory (the default is: C:\Program Files\Informix\Formation\Samples\newshoppoos.txt).

The PromoMap Data Store

This is a two-column text file that maps the outdated promotion keys used by Store 1 to the corporate standard keys used by Store 2 and in the corporate database.

Field	Description	Data Type
oldkey	Outdated promotion code used by Old Shop	Integer
newkey	Corporate standard promotion key	Integer

The file that stores the Promomap data is located in the Samples directory under the main Formation directory (the default is: C:\Program Files\Informix\Formation\Samples\promomap.txt).

Period Table

Each date in a sales transaction is mapped to a unique key before being stored in the Sales table, so each record must have the date replaced by the key from the Period table.

Field	Description
perkey	Integer that identifies exactly one row in the Period table
date	Date value that identifies each day in the format YYYY-MM-DD
day	Character-string abbreviation of the day of the week. Examples: SA, SU, MO
week	Integer that identifies each week of each year by number, with each week starting on a Sunday
month	Character-string abbreviation of the name of each month. Examples: JAN, FEB, MAR
qtr	Character string that uniquely identifies each quarter. Examples: Q1_94, Q3_95
year	Integer that identifies the year. Examples: 1994, 1995, 1996

This data store and the Sales Table data store were both imported from the Aroma database, using the Data Definition Wizard.

Sales Table

The Aroma Sales table consists of sales for each day, by product, store, and promotion code. The first five fields form a multi-part primary key.

Field	Description
perkey	Key to the Period Table (one key for each unique date)
classkey	Key to Class Table
prodkey	Key to Product Table
storekey	Key to Store Table
promokey	Key to Promotion Table
quantity	Amount of sales per day
dollars	Dollars per day

Setting Up Import and Export Operators

The data flow diagram uses the import and export operators described in the following table.

Name in Data Flow Diagram	Operator and Description
OldShop_POS	File Import Reads the data from the oldshoppo.txt file and creates a data flow containing all records in that text file.
Promomap	File Import Reads the data from the promomap.txt file and creates a data flow containing all records in that text file.
NewShop_POS	File Import Reads the data from the newshoppo.txt file and creates a data flow containing all records in that text file.
Aroma_Period_Table	Red Brick Import Reads data from the Period table in the Aroma Red Brick Warehouse database to the input data flow.
Add_to_Aroma_Sales_table	Red Brick Export Writes data from the data flow to the Sales table in the Aroma Red Brick Warehouse database.
Rejected_records	File Export Writes records with dates that did not match dates in the Period table to a file for later cleanup.

The OldShop_POS File Import Operator

The File Import operator named OldShop_POS is set up to read sales data for the store from the file oldshoppoos.txt. The parameter values specified for this operator are:

Parameter Values

Record Definition: OldShop data store

File List: oldshoppoos.txt in the Formation Samples directory.

Important: You may need to edit this location if you installed the Formation Architect in a location other than the default (C:\Program Files\Informix\Formation).



The Promomap File Import Operator

The File Import operator named Promomap contains promotion codes used by the old shop and their corresponding corporate promotion codes. The parameter values for this operator are:

Parameter Values

Record Definition: PromoMap data store

File List: promomap.txt in the Formation Samples directory.

Important: You may need to edit this location if you installed the Formation Architect in a location other than the default (C:\Program Files\Informix\Formation).



The NewShop_POS File Import Operator

The File Import operator, NewShop_POS, is set up to read sales data for the store from the file newshoppoos.txt.

Parameter Values

The required parameters you must specify are set to the following values:

Record Definition: NewShopPOS data store

File List: newshoppoos.txt in the Formation Samples directory.



Important: You may need to edit this location if you installed the Formation Architect in a location other than the default (C:\Program Files\Informix\Formation).

The Aroma_Period_Table Red Brick Import Operator

The Red Brick Import operator, `Aroma_period_table`, is set up to read data from the Period table in the Aroma database. The required parameters you must specify for this operator are set to the following values:

Parameter Values

Table Definition: Period data store
Table Name: period
User Name: system
Password: manager

The Add_to_Aroma_sales_table Red Brick Export Operator

The Red Brick Export operator, `Add_to_Aroma_sales_table`, is set up to write the data at the end of the data flow to the Sales table in the Aroma database. The required parameters you must specify are set to the following values:

Parameter Values

Table Definition: Sales data store
Table Name: sales
User Name: system
Password: manager
Load mode: modify

The Rejected_records File Export Operator

The File Export operator named `Rejected_records` writes records with date values that did not match any date values in the Period table to a file for later processing.

Parameter Values

Record Definition: NewShopPOS
File List: C:\docsamp2\rejects.txt
Overwrite: True

Transforming the Data

The job performs a series of transformations, as explained in the following sections:

- [Cleaning Up the OldShop Data](#)
- [Cleaning Up Rejected Records in the OldShop Data Flow](#)
- [Combining the Two OldShop Data Flows](#)
- [Creating a Union of the Data Flows](#)
- [Adding the Period Key](#)
- [Grouping the Data by Key](#)

Cleaning Up the OldShop Data

This section explains the following transformations on the OldShop data:

- Replaces the old promotion code with the corporate code in each record
- Adds the store key to each record
- Reorders the fields to match the fields in the NewShop data definition

All these steps can be performed by using the Join operator, described as follows.

Name in Data Flow Diagram	Operator and Description
Clean_up_data	Join operator Combines fields from the OldShop data flow and the Promomap data flow to create a new data flow with a new output data definition.

Input and Output Data Definitions

To begin the setup of the Join operator, the two data flows are connected to the two input ports of the operator as follows:

- The data flow from the OldShop import operator is connected to the Left (large) input port.
- The data flow from the Promomap import operator is connected to the Right (small) input port.

Defining the Output Data Definition on the MatchedPair Port

The data definition on the Matched Pair output port of the Join operator (Clean_up_data) was edited:

- Removed the Transaction and Promotion Code fields from the OldShop input data flow and removed the oldkey field from the promomap input data flow
- Added a new Store Key field to store the corporate store number
- Reordered the fields to match the NewShop data definition

Selecting Keys

The Join operator requires a key for each table that is used to match records, so the job uses Date as the key for both the Left key (the OldShop data flow) and the Right key (the promomap data flow).

Projecting the Input Data to the Output Data Definition

In the MatchedPair Project parameter of the Join operator, the following was done:

- A constant with a value of 10 was added to represent the Store key.
- The fields from the data flows attached to the right and left input ports were mapped to fields in the output data definition, as shown in the snippet window below.

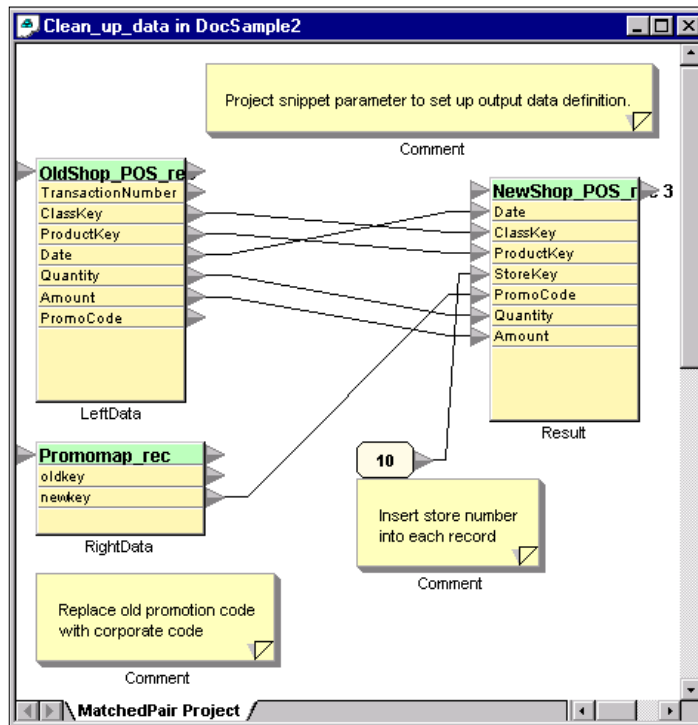


Figure A-2
Project Snippet

Cleaning Up Rejected Records in the OldShop Data Flow

Any records from the OldShop data flow that do not have a match for a promotion code in the promokey data flow are directed to the UnmatchedLeft output port of the Join operator. This data flow is then attached to a Project operator.

Name in Data Flow Diagram	Operator and Description
Add_default_value_for_promo_code	Project Operator Replaces promotion code with a default code indicating no promotion and adds in the store key

In the Project parameter, the following was done:

- A constant with a value of 10 was added to represent the Store key.
- The promotion code was replaced with the integer value 1 to indicate the standard coupon promotion was in force when the sale was made.

- The fields from the data flows attached to the right and left input ports were mapped to fields in the output data definition, as shown in the snippet window below.

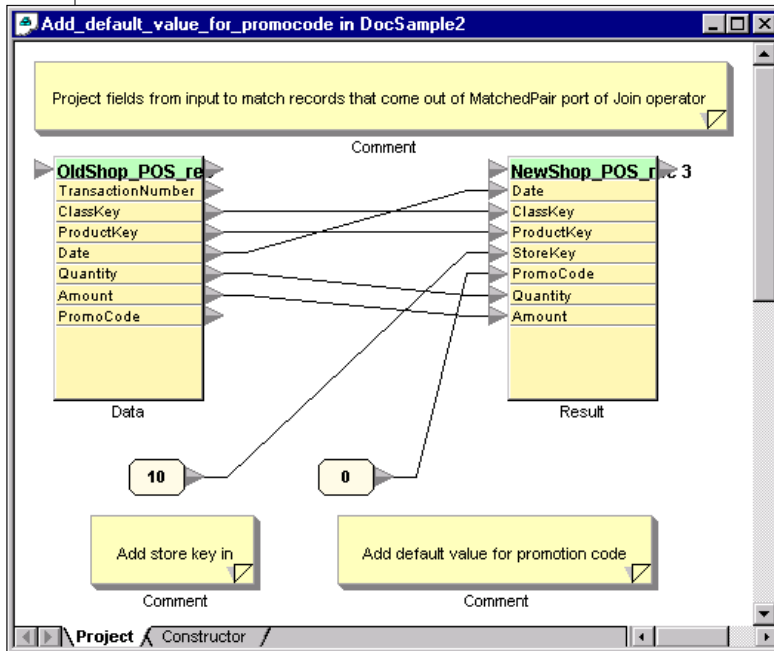


Figure A-3
Mapping Right and
Left to the Output
Data Definition

Combining the Two OldShop Data Flows

After the rejected records have been cleaned up, the job merges those records back into the main data flow for the OldShop records. To perform the merge, the job uses the Union operator.

Name in Data Flow Diagram	Operator and Description
Union_both_data_flows_from_OldShop	Union Operator
	Merges two data flows with identical structures

Setting up the Input Ports

To set up the Union operator, the two data flows are connected to the two input ports of the operator as follows:

- The data flow from the MatchedPair Project of the Join operator is connected to Data(0).
- The data flow from the output port of the Project operator is connected to Data(1).

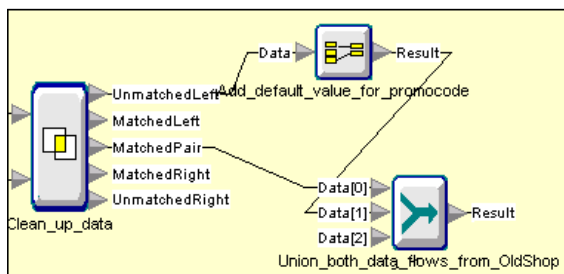


Figure A-4
*The Union Operator
in a Data Flow
Diagram*

Since the Union is not using a key and the job takes the default output order, there are no parameters that need to be specified.

Creating a Union of the Data Flows

Now that the OldShop data has been cleaned up and formatted to match the NewShop data, the job brings in the NewShop data and performs a union of the two data flows.

Name in Data Flow Diagram	Operator and Description
Union_Shop_Data	Union Operator Merges two data flows with identical structures

Setting up the Input Ports

To set up the Union operator, the two data flows are connected to the two input ports of the operator as follows:

- The NewShop data flow is connected to Data(0).
- The OldShop data flow is connected to Data(1).

Because the Union is not using a key and the job takes the default output order, no parameters need to be specified.

Adding the Period Key

After the data from both stores has been combined into one data flow, the data flow diagram then uses a Join operator to replace the Date value in each record with the associated Period key from the Aroma Period table. These records will be output on the MatchedPair output port.

In addition, any records with a Date value that do not match a Date value in the Period table will be output on the UnmatchedLeft output port.

Name in Data Flow Diagram	Operator and Description
Lookup_and_replace_Date	Join (equi-join operation) on the Date fields

Input and Output Data Definitions

To begin the setup of the Join operator, the two data flows are connected to the two input ports of the operator as follows:

- The data flow from the store import operator is connected to the Left (large) input port.
- The data flow from the Period import operator is connected to the Right (small) input port.

The data definition on the MatchedPair output port was edited to remove the Date field and replace it with the integer field named perkey.

The data definition on the UnmatchedLeft output port did not need any editing.

Left Key and Right Key Parameters

Both inputs to the Join use Date as the key.

The MatchedPair Project Parameter

The fields from the data flows attached to the LeftData and RightData ports were mapped to fields in the output data definition, as shown in the following snippet window.

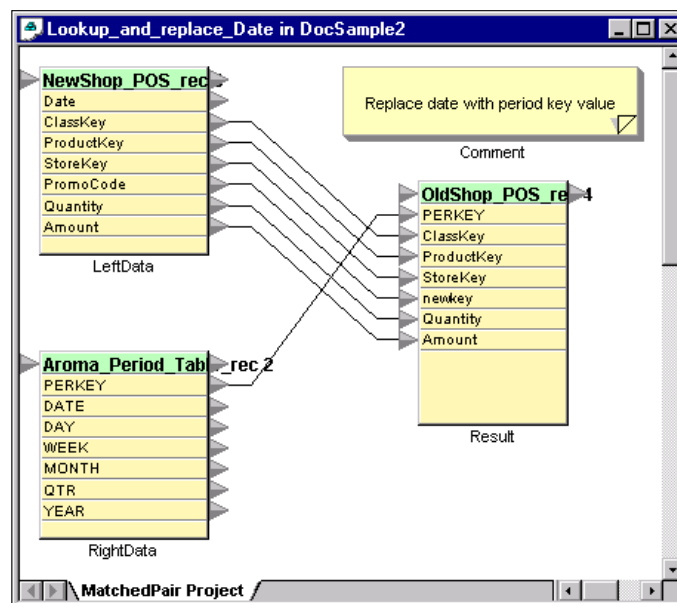


Figure A-5
Completing the
MatchedPair Project

Grouping the Data by Key

Finally, the data flow diagram groups records by a five-part key into summary records for each key, then totals the Quantity and Dollar amounts for each summary record.

To start the grouping, the data flow from the MatchedPair output port of the Join operator is connected to the Data input port on the Group By operator.

Name in Data Flow Diagram	Operator and Description
Group_By_Multipart Key	Group By Collects the input data flow by one or more key fields, aggregates the data by the one or more key fields, and writes the summarized data to the output data flow

The Group By operator set up is explained in the following sections.

Selecting the Keys

The Sales table stores the sales data using a multi-part key: period, class, product, store, and promotion code. All five fields are selected in that order as the key to use in grouping the records.

Defining the Summary Record

Next, the fields to be summarized are specified as the summary record. The Quantity field (an integer) and the Amount field (decimal, with a precision of 2) are the two fields in this summary record. Think of the summary record as a running total of the calculations done by the Group By operator.

Initializing the Summary Record

Each time the Group By operator finds a new key, all fields in the summary record are initialized to zero in the Initialize snippet parameter.

Because the Formation Architect supports automatic data conversion whenever possible, a single integer value of 0 (zero) is converted to 0.0 when connected to the decimal Amount field. The circle on the line between the constant and the Amount field denotes an automatic conversion.

Mapping the Input to the Summary Record

The Map snippet parameter maps the fields in the input data definition to the fields defined in the summary record. In the sample, there is a simple one-to-one mapping between the Quantity and Amount fields, indicated by the lines connecting the fields in the Data block to fields in the summary record.

Calculating the Summaries

The Aggregate snippet parameter is where calculations take place.

The two SummaryRecord blocks are temporary storage that hold all the records that are being aggregated by the operator. To produce the final, grouped record for the entire group of records, you combine the two SummaryRecord blocks into the SummaryRecordResult block.

In this job, the Quantity fields are added together and stored in the Quantity field of the SummaryRecordResult block. In the same fashion, the Amount fields are added together and stored in the Amount field of the SummaryRecord Result block.

Finalizing the Output Record

When all the records with a specific key have been processed, the final summary record is ready to be written to the output record.

To create the Result block for this snippet, the data flow from the input port of the next operator, Red Brick Export, is connected to the output port of the Group By operator. This action automatically propagates the data definition from the export operator to the Group By, ensuring that the data flow produced by the Group By operator matches the format of the data flow required by the export operator.

The Finalize snippet parameter maps the fields in the summary record to fields in the output record, and the fields used as keys are also mapped to output record fields.

Exporting the Data

The transformed records can now be output to the Aroma Sales table by using the Red Brick Export operator and selecting the Aroma Sales table data store as the value of the Table Definition parameter.

Name in Data Flow Diagram	Operator and Description
Add_to_Aroma_Sales_table	Red Brick Export Writes data from the data flow to a Red Brick Warehouse database table

Setting up the Export Parameters

The Red Brick Export operator is set up to write data to the Sales table in the Aroma database. The required parameters for this operator are set to the following values:

Table Definition: Sales data store
Table Name: sales
User Name: system
Password: manager
Load mode: modify

Running the Job

Now that the data flow diagram is finished, you can validate the job, then generate the source files, build the executable, and run the job.

For more information on generating , building, running and monitoring the job, refer to [Chapter 2, “DocSample1: A Quick Start Guide.”](#)

Summary of Supported Data Types

This chapter lists the data types that are supported by Informix Formation. Each section is for a different data store and lists the supported data types, and where appropriate, the range of values for the precision, scale, or length of the fields for the data type. This chapter is organized into the following sections:

- [File Database Data Types](#)
- [Microsoft SQL Server Data Types](#)
- [Oracle Server Data Types](#)
- [Red Brick Warehouse Data Types](#)
- [Informix Dynamic Server and Informix Dynamic Server/AD Data Types](#)
- [ODBC Data Types](#)

File Database Data Types

The following is a list of the supported data types for file databases. A checkmark (✓) in the table cell indicates that the data type is supported in the listed field type.

The accepted length for all fixed length fields is any integer value greater than 0.

Data Type	Delimited Flat Files or Text Files	Fixed Length Text File	Fixed Length Flat File
BLOB			✓
Date	✓	✓	
Decimal	✓	✓	
Double	✓	✓	✓
Float	✓	✓	✓
Integer	✓	✓	✓
Interval	✓	✓	
Long	✓	✓	✓
Short	✓	✓	✓
Text	✓	✓	✓
Time	✓	✓	
Timestamp	✓	✓	
Unsigned Integer	✓	✓	✓
Unsigned Long	✓	✓	✓
Unsigned Short	✓	✓	✓

Microsoft SQL Server Data Types

The following table lists the supported data types for the Microsoft SQL Server database. Restrictions to the length, precision or scale of these data types, if any, are also listed.

Data Type	Length Value	Precision Value	Scale Value
binary	1 to 255	—	—
bit	—	—	—
char	1 to 255	—	—
datetime	—	—	—
decimal	—	1 to 38	0 to precision value
float	—	1 to 53	—
image (See Note)	—	—	—
int	—	—	—
money	—	—	—
numeric	—	1 to 38	0 to precision value
real	—	—	—
smalldatetime	—	—	—
smallint	—	—	—
smallmoney	—	—	—
text (See Note)	—	—	—
timestamp	—	—	—
tinyint	—	—	—
varbinary	1 to 255	—	—
varchar	1 to 255	—	—



Important: Although you can define a table that contains image or text data types, the use of these data types in a data flow diagram is not supported, and fields with this data type are ignored. You should drop those fields or change the fields to a supported data type.

Oracle Server Data Types

The following table lists the supported data types for the Oracle Server database. Restrictions to the length, precision or scale of these data types, if any, are also listed.

You can determine the precision and scale of the data types in your Oracle Server database using the Oracle Schema Manager. Use the Oracle Schema Manager to view the fields in the database table to be brought into Formation. In the Column Details box, the Length area lists the Formation precision value, and the Precision area lists the Formation scale value.

Data Type	Length Value	Precision Value	Scale Value
char	1 to 255	—	—
date	—	—	—
float	—	Binary precision has a value between 1 to 126	—
long (See Note)	1 to 999999	—	—
long raw (See Note)	1 to 999999	—	—
mlslabel (See Note)	—	—	—
number (See Note)	—	1 to 38 or NULL	-84 to 127 or NULL; default is NULL
raw	1 to 255	—	—
rowid (See Note)	—	—	—
varchar2	1 to 2000	—	—



Important: Although you can define a table that contains the following data types, the use of these data types in a data flow diagram is not supported, and fields with that data type are ignored. You should drop those fields or change the fields to a supported data type. The data types that can be defined, but not used are: long, long raw, mlslabel, rowid, and number (when number has a negative scale).

Red Brick Warehouse Data Types

The following table lists the supported data types for the Red Brick Warehouse database. Restrictions to the length, precision or scale of these data types, if any, are also listed.

Data Type	Length Value	Precision Value	Scale Value
Char	1 to 1024, default is 1	—	—
Character	1 to 1024, default is 1	—	—
Date	—	—	—
Dec	—	1 to 38, default is 9	0 to precision value, default is 0
Decimal	—	1 to 38, default is 9	0 to precision value, default is 0
Double Precision	—	—	—
Float	—	—	—
Int	—	—	—
Integer	—	—	—
Num	—	1 to 38, default is 9	0 to precision value, default is 0
Numeric	—	1 to 38, default is 9	0 to precision value, default is 0

(1 of 2)

Data Type	Length Value	Precision Value	Scale Value
Real	—	—	—
Smallint	—	—	—
Time	—	—	0 to 6, default is 0
Timestamp	—	—	0 to 6, default is 0
Tinyint	—	—	—

(2 of 2)

Informix Dynamic Server and Informix Dynamic Server/AD Data Types

The following table lists the supported data types for the Informix Dynamic Server and Informix Dynamic Server/AD databases. Restrictions to the length, precision or scale of these data types, if any, are also listed.

Data Type	Length Value	Precision Value	Scale Value
Byte	2 to 2147483646, default is 12	—	—
Char	1 to 32767, default is 1	—	—
Character	1 to 1024, default is 1	—	—
Date	—	—	—
Datetime, excluding Fractions	—	---	---
Datetime Year to Fraction	—	—	1 to 5, default is 3
Datetime Month to Fraction	—	—	1 to 5, default is 3

(1 of 3)

Data Type	Length Value	Precision Value	Scale Value
Datetime Day to Fraction	—	—	1 to 5, default is 3
Datetime Hour to Fraction	—	—	1 to 5, default is 3
Datetime Minute to Fraction	—	—	1 to 5, default is 3
Datetime Second to Fraction	—	—	1 to 5, default is 3
Datetime Fraction to Fraction	—	—	1 to 5, default is 3
Decimal	—	1 to 32, default is 16	0 to precision value, default is 0
Float	—	—	—
Integer	—	—	—
Interval Year to Year	—	1 to 9, default is 4	—
Interval Year to Month	—	1 to 9, default is 4	—
Interval Month to Month	—	1 to 9, default is 2	—
Interval Day to Day	—	1 to 9, default is 2	—
Interval Day to Hour	—	1 to 9, default is 2	—
Interval Day to Minute	—	1 to 9, default is 2	—
Interval Day to Second	—	1 to 9, default is 2	—
Interval Day to Fraction	—	1 to 9, default is 2	1 to 5, default is 3

(2 of 3)

Data Type	Length Value	Precision Value	Scale Value
Interval Hour to Hour	—	1 to 9, default is 2	---
Interval Hour to Minute	—	1 to 9, default is 2	---
Interval Hour to Second	—	1 to 9, default is 2	---
Interval Hour to Fraction	—	1 to 9, default is 2	---
Interval Minute to Minute	—	1 to 9, default is 2	---
Interval Minute to Second	—	1 to 9, default is 2	---
Interval Minute to Fraction	—	1 to 9, default is 2	1 to 5, default is 3
Interval Second to Second	—	1 to 9, default is 2	---
Interval Second to Fraction	—	1 to 9, default is 2	1 to 5, default is 3
Interval Fraction to Fraction	—	---	1 to 5, default is 3
Money	---	1 to 32, default is 16	0 to precision value, default is 2
Nchar	1 to 32767	---	---
Nvarchar	1 to 255	---	---
Serial	---	---	---
Small Float	---	---	---
Small Int	—	—	—
Text	2 to 2147483646	—	---
Varchar	—	—	---

(3 of 3)

ODBC Data Types

The following table lists the supported ODBC data types. Restrictions to the length, precision or scale of these data types, if any, are also listed.

Data Type	Length Value	Precision Value	Scale Value
Char	1 to 65535, default is 1	—	—
Varchar	1 to 65535, default is 1	—	—
Longvarchar	1 to 2147483647, default is 1	—	—
Decimal	—	0 to 999999, default is 38	0 to precision value, default is 0
Numeric	—	0 to 999999, default is 38	0 to precision value, default is 0
Bit	—	—	—
Tinyint	—	—	—
Smallint	—	—	—
Integer	—	—	—
Bigint	—	20	—
Real	—	—	—
Float	—	—	—
Double	—	—	—
Binary	1 to 65535, default is 1	—	—
Varbinary	1 to 65535, default is 1	—	—

(1 of 2)

Data Type	Length Value	Precision Value	Scale Value
Longvarbinary	1 to 2147483647, default is 1	—	—
Type Date	---	—	—
Type Time	---	—	—
Type Timestamp	---	—	—
Interval Data Types	—	---	—
Date	—	---	—
Time	—	---	—

(2 of 2)

Glossary

code snippet	One or more functions inside a parameter that represents some unit of processing specified by C++ code.
Comment function	Creates a note area in the job or visual snippet for entering descriptive text or comments. The default style is Comment, with the icon displayed as yellow. You can change the style to Bugs (pink icon) or Todo (green icon).
Custom folder	The folder on the Toolbox tab of the Navigator that presents any custom items the user has created in the current workspace. Contrast with Standard folder.
custom function	A function that a user creates and saves, such as an operator variable or an external function. Stored in the Custom folder of the Toolbox tab of the Navigator.
custom object	An object that is created by the user, such as a custom function, external function, or operator variable.
data definition	The structure (records for a file, tables for a database) of the data flow as defined on input and output ports of operators in a data flow diagram. The user can view the data definition for a specific operator port by double-clicking on that port; this opens the Data Definition Editor dialog (if the data definition can be edited) or a read-only Data Definition dialog (if the user cannot edit it at this port).
data definition block	In a parameter snippet for an operator, the yellow boxes that list the columns or fields for the input and output data definitions used in the snippet. Usually these blocks are named Data and Result.

Data Definition Editor	The dialog that enables a user to edit the data definition for an operator port in the data flow diagram. Double-click on the port to open this dialog. If the data definition cannot be edited, you see a read-only version of this dialog.
Data Definition Wizard	The series of dialogs that enable a user to import schemas and define and edit data stores and their contents (tables, records, fields).
data flow	The processing path data takes from one operator to another. Data flow is shown graphically in a job diagram by drawing a line between an input port and an output port.
data flow diagram	The visual representation of the processing to be performed on a set of data. It consists of operators connected by data flows. The information in the data flow diagram is used to generate job executables that actually perform the processing.
data source	In the schema importer, the ODBC data source to use for importing schema.
data store	A collection of related tables or records. A data store can refer to either a relational database or a file. It contains one or more table definitions (for a relational database) or record definitions (for a file).
data tab	The rightmost tab in the Navigator; it displays data stores defined for the workspace.
data transformation	Process of sorting, grouping, and removing errors and inconsistencies from data being imported into a data warehouse. This process includes checking for adherence to standards, internal consistency, referential integrity, valid domain, and includes replacing and repairing incorrect data with correct data.
declaration	The parameter available in some operators where the user can specify external calls, such as #define and #include.
export operator	An operator that loads transformed data from a data flow into a table or flat file.
external function	A library or executable that can be called by a snippet. Defined by the user and available by default from the Custom folder.
external library	A software program that is available for use with certain operators on the processors on which the software program is licensed and installed.
field	The components of a table or record.

Flow Engine machine	The machine where the data transformations actually occur.
Formation Architect	The GUI portion of Formation, where the user creates job diagrams and generates job executables.
Formation Flow Engine	The back-end portion of Formation, where job executables are run.
Formation Log Utility	The Formation component that allows you to collect information about the execution of a job. The Log Utility is similar to the Red Brick Warehouse logging subsystem. You use the Formation Log Viewer to view this information.
Formation Log Viewer	The Formation component that allows you to view information about the execution of a job. The Log Viewer is similar to the Red Brick Warehouse log viewer. You use the Formation Log Utility to collect information to use with the Log Viewer.
input port	The point on an operator where data enters the operator for processing.
input variable	In a snippet, the icon that represents data entering the snippet for processing.
job	The set of data flow diagrams and executables that work together to define and create the contents of a data warehouse. Includes (but not limited to) configuration information, schema information, data flow diagrams, and job executables.
job executables	The set of executable files generated by the data flow diagram.
Job tab	The middle tab of the Navigator; it displays a tree list view of the operators and functions in the active job.
key	One or more fields in a data definition that can be used as the basis for operations such as sort, compare, or filter.
log file	The file to which messages are written by the Formation Log Utility.
Navigator	The three-tabbed palette that displays operators and function information, job information, and database information. The three tabs are: Toolbox, Job, and Data Stores.
operator	A representation of some data processing activity, such as sorting or exporting data. An operator has one or more ports connected to data flows. The operator also has parameters that allow the user to control various attributes of the operator.

operator variable	A variable that can be used more than once in an operator. Defined by the user and available from the Custom folder.
ordering links	In visual snippets, the links that can be used for scheduling the order of processing.
output port	The point on an operator where the data flow leaves the operator for the next stop in the data flow.
output variable	In a snippet, the icon that represents data exiting the snippet after processing.
package	A grouping of functions and variables that a user places inside the package window to hide the complexity of a snippet.
parameter	Information about an operator or port that is specified by the user. Parameters can be as simple as numeric or string values, or they can be as complex as C++ functions.
parameter dialog	The table where a user can list and edit the parameters for an operator.
port	The point at which a data flow is connected to an operator. The number of input and output ports available is defined for each operator.
precision	Number of digits of accuracy. For example, the number 123.45 has a precision of 5.
record	A sublevel in a data store that defines a text file or flat file. Below this level are fields, analogous to fields in a text file.
runtime parameters	The set of parameters used when the job is processed by the Formation Flow Engine.
runtime parameter files	The files generated as part of a job that contain the runtime parameters for a job. These files are named rbf_profile.dat and rbf_machine.dat.
scale	Number of digits to the right of the decimal place. For example, the number 123.45 has a scale of 2.
snippet	A kind of parameter that requires the user to provide programmatic logic using either a visual snippet or a code snippet.
Standard folder	The folder on the Toolbox tab of the Navigator that presents the operators and functions shipped with the Formation product. Contrast with Custom folder.

standard functions	The set of functions shipped with the Formation software.
standard operators	The set of operators shipped with the Formation software.
table	A sublevel in a data store, analogous to a table in a database. A data store can contain many table definitions.
temporary space	Disk space for temporary use by one or more processes.
title bar	The top line of a window, where the window title and other (optional) context information—such as job name or workspace—is displayed.
toolbar	The toolbar displayed in the Architect window.
Toolbox palette	Optional pane at the bottom of the Architect window that displays icons for the currently selected folder on the Job tab of the Navigator.
Toolbox tab	The leftmost tab in the Navigator; it displays the standard operators and functions and all user-defined functions in the current workspace.
transformation operators	The set of operators that transform data, such as the Join and Project operators.
validation	The process of checking a job to make sure all required operator parameters are specified, all required input and output ports are connected, and snippet parameters have been specified.
Visual Snippet	One or more functions inside a parameter that visually represents some unit of processing, such as a data type conversion.
workspace	A container for one or more jobs and one or more data stores. Workspaces can be used to organize jobs for different groups, departments, or companies.
Workspace Manager	The utility where a user can add, delete, import and export workspaces.

Index

A

Advanced Join operator 6-25
Aggregate operator 6-29
Architect 1-7
Artificial keys 6-37

C

Cartesian product join 6-27
Code page 4-13
Code snippets 7-21
Comment function 7-8
Components of Formation 1-4
Conditional functions 7-10
Cross Product Join operator 6-27
Cursor operator 6-30
Custom functions 7-14

D

Data
 copying between NT and UNIX 4-17
 creating 4-8
 defining 4-3
 English 4-11
 importing schema for 4-10
 Japanese 4-11
 locales in 4-11
 modifying 4-8, 4-10
 multibyte 4-11
 multibyte text data type 4-14
 naming for 4-9
 normalizing 6-31
 propagation in 5-9

 sources 4-3
 specifying location of 5-7
 synchronizing 6-10
 using data stores with 4-5
Data Definition Wizard 4-8
Data flow diagrams
 connecting operators in 5-8
 defining data stores for 5-6
 introduction to 1-9
 modifying fields in 4-10
 operator parameters in 5-10
 overview of 5-3
 propagation of data in 5-9
 selecting operators for 5-6, 5-7
 specifying data location for 5-7
 specifying data store for 5-7
 steps in creating 5-4
 validating 5-11
Data propagation 5-9
Data stores 4-5
 defining 5-6
 locales in 4-12
 specifying 5-7
Data types
 in visual snippets 7-16
 multibyte 4-11, 4-14
 supported B-1
Deduplicate operator 6-20
DocSample1 sample job 2-3
DocSample2 sample job A-1

E

Environment variables
 LD_LIBRARY_PATH 9-10
 RBF_BUILD_OPTION 9-10

RBF_CONFIG 9-10
 RBF_HOME 9-11
 RBF_HOST 9-11
 RBF_ORACLE 9-11
 RBF_TEMP 8-10
 Event logging
 categories 9-25, 9-26
 log daemon 9-17
 log files 9-26
 severity levels 9-26
 Example jobs
 DocSample1 2-3
 DocSample2 A-1
 Executable master file 2-23
 Execution order parameter 6-10
 Export operators 6-8
 Exporting workspaces 3-7
 External function 10-7

F

File Export operator 6-12
 File Import operator 6-11
 Filter operator 6-21
 First Normal operator 6-31
 Flow Engine 1-11
 Flow Engine machines
 configuring 8-7
 creating 8-5, 8-6
 deleting 8-6
 local machine 8-4
 locales in 4-12
 modifying 8-5, 8-6
 overview of 8-3
 temporary space for 8-8
 Formation Architect 1-7
 Formation components 1-4
 Formation Flow Engine 1-11
 Functions
 Comment 7-8
 conditional 7-10
 custom 7-14
 external 10-7
 If Then Else 7-10
 Package 7-9
 standard 7-7
 syntax 7-8
 to hide complexity 7-9

G

Gather operator 6-39
 Generated files
 rbf_job 9-12
 rbf_machine.dat 9-6, 10-3
 rbf_profile.dat 9-6, 10-4
 Group By operator 6-32

H

Household operator 6-33

I

If Then Else function 7-10
 Import operators 6-8
 Importing
 schema 4-10
 workspaces 3-4
 Informix DS AD Export
 operator 6-15
 Informix DS AD Import
 operator 6-14
 Informix DS Export operator 6-13
 Informix DS Import operator 6-13
 Informix Dynamic Server 6-13
 Informix Dynamic Server/
 AD 6-14, 6-15
 Integrating external
 applications 6-34
 Intersolv ODBC 6-17

J

Japanese data 4-11
 Jobs
 creating 3-11
 defining temporary space for 8-8
 deleting 3-11
 DocSample1 example 2-3
 DocSample2 example A-1
 duplicating 3-11
 managing 3-10
 opening 3-11
 selecting operators for 5-6
 specifying data store for 5-7

troubleshooting 9-13, 9-30
 validating 5-11
 Join operator 6-25
 Join operators
 Advanced Join 6-25
 Cross Product Join 6-27
 Join 6-25

L

LD_LIBRARY_PATH environment
 variable 9-10
 Local machine 8-4
 Locales for data 4-11
 Log files 9-26
 naming conventions 9-26
 removing 9-27
 specifying location 9-28
 specifying size 9-28
 Log message categories 9-25
 Log service on Windows NT 9-19
 Log utility
 configuration of 9-26
 files from 9-26
 on NT 9-19
 on UNIX 9-17
 removing log files 9-27
 Log viewer 9-21

M

Master executable file 2-23, 9-12
 Message categories 9-25
 Message severity levels 9-26
 Microsoft SQL Server 6-16
 MSSQL Server Export
 operator 6-16
 MSSQL Server Import
 operator 6-16
 Multibyte data 4-11

N

Normalizing data 6-31
 Null values 7-17

O

ODBC Export operator 6-17
 ODBC Import operator 6-17
 Operator parameters 5-10, 6-7
 Operator ports 6-6
 Operator variable 7-15
 Operators
 Advanced Join 6-25
 Aggregate 6-29
 connecting 5-8
 Cross Product Join 6-27
 Cursor 6-30
 Deduplicate 6-20
 File Export 6-12
 File Import 6-11
 Filter 6-21
 First Normal 6-31
 Gather 6-39
 Group By 6-32
 Household 6-33
 import and export 6-8
 Informix DS AD Export 6-15
 Informix DS AD Import 6-14
 Informix DS Export 6-13
 Informix DS Import 6-13
 Join 6-25
 MSSQL Server Export 6-16
 MSSQL Server Import 6-16
 Null values in 7-17
 ODBC Export 6-17
 ODBC Import 6-17
 Oracle Server Export 6-18
 Oracle Server Import 6-18
 parameters for 5-10, 6-7
 Partition 6-38
 ports 6-6
 Program 6-34
 Project 6-22
 Red Brick Export 6-19
 Red Brick Import 6-19
 selecting import and export
 operators 5-6
 selecting transformation
 operators 5-7
 Sort 6-22
 Split 6-36
 summary of import and
 export 6-10

Surrogate Key 6-37
 Union 6-23
 Oracle Server Export operator 6-18
 Oracle Server Import operator 6-18
 Ordering links 7-6

P

Package function 7-9
 Parallel processing 1-11
 Parameters of operators 6-7
 Partition operator 6-38
 Ports of operators 6-6
 Program operator 6-34
 Project operator 6-22

R

rbflogd daemon 9-17
 rbflogview executable 9-21
 RBF_BUILD_OPTION
 environment variable 9-10
 RBF_CONFIG environment
 variable 9-10
 RBF_HOME environment
 variable 9-11
 RBF_HOST environment
 variable 9-11
 rbf_job master executable file 2-23,
 9-12
 rbf_machine.dat generated file 9-6,
 10-3
 RBF_ORACLE environment
 variable 9-11
 rbf_profile.dat generated file 9-6,
 10-4
 RBF_TEMP environment
 variable 8-10
 Red Brick Export operator 6-19
 Red Brick Import operator 6-19

S

Sample jobs
 DocSample1 2-3
 DocSample2 A-1
 Schema importing 4-10

Snippets

 code 7-21
 visual 7-3
 Sort operator 6-22
 Split operator 6-36
 SQL Server 6-16
 Standard functions 7-7
 Surrogate Key operator 6-37
 Synchronizing data 6-10
 Syntax functions 7-8

T

Temporary space
 default locations for 8-10
 for jobs 8-8
 guidelines for setting 8-11
 Troubleshooting jobs 9-13, 9-30

U

Union operator 6-23

V

Validating 5-11
 Variables
 in visual snippets 7-6
 operator variable 7-15
 Viewer for log files 9-21
 Visual snippets
 custom functions 7-14
 data types 7-16
 defining 7-5
 Null values in 7-17
 operator variables in 7-15
 ordering links in 7-6
 overview of 7-3
 standard functions 7-7
 syntax functions 7-8
 variables in 7-6

W

Workspace
 backup of 3-7
 copying 3-4

creating 3-4
 deleting 3-9
 exporting 3-7
 importing 3-4
 locales in 4-12
 opening 3-7
 overview of 3-3
 renaming 3-9