Rapid Mart Development Guide

Rapid Mart 2.1

Contents

Introduction

chapter 1

About this guide

The Rapid Mart Development Guide contains the technical information you need to develop and maintain a Rapid Mart. This document provides an overview of BusinessObjects products, discusses where Rapid Marts™ fit in the BusinessObjects architecture, and describes the Rapid Mart development process. This document includes programming conventions to follow and advanced design issues to consider when developing a Rapid Mart.

This chapter describes the manual and how you can best use the manual.

Who should read this guide

This document provides information about designing and developing a Rapid Mart using Data Integrator. The information is intended to help:

- System integrators
- BusinessObjects consultants
- Data Integrator users

This document assumes that you have technical knowledge of databases and the back office or source system from which you will be extracting data, and that you are about to develop a Rapid Mart using Data Integrator. If you have never used Data Integrator™ or Rapid Marts™, you should attend one or more of the education classes that BusinessObjects offers. Contact your BusinessObjects representative to learn about the current education schedule.

Business Objects information resources

Other reference materials that you might find useful include:

- Data Integrator Getting Started Guide
- Data Integrator Designer Guide
- Data Integrator Administrator Guide
- Data Integrator Reference Guide
- Data Integrator Supllement for SAP R/3

For more information and assistance, see Appendix C: Business Objects Information Resources. This appendix describes the BusinessObjects documentation, customer support, training, and consulting services, with links to online resources.

Overview

chapter 2

BusinessObjects Data Platform

The BusinessObjects Data Platform is a set of applications from BusinessObjects Data Integration, Inc., that enterprises can use to manage all types of data—from data for supply chain partners to data for customers. The Business Objects Data Platform packages and speeds the delivery of ERP, SCM, and CRM data for analysis across the enterprise.

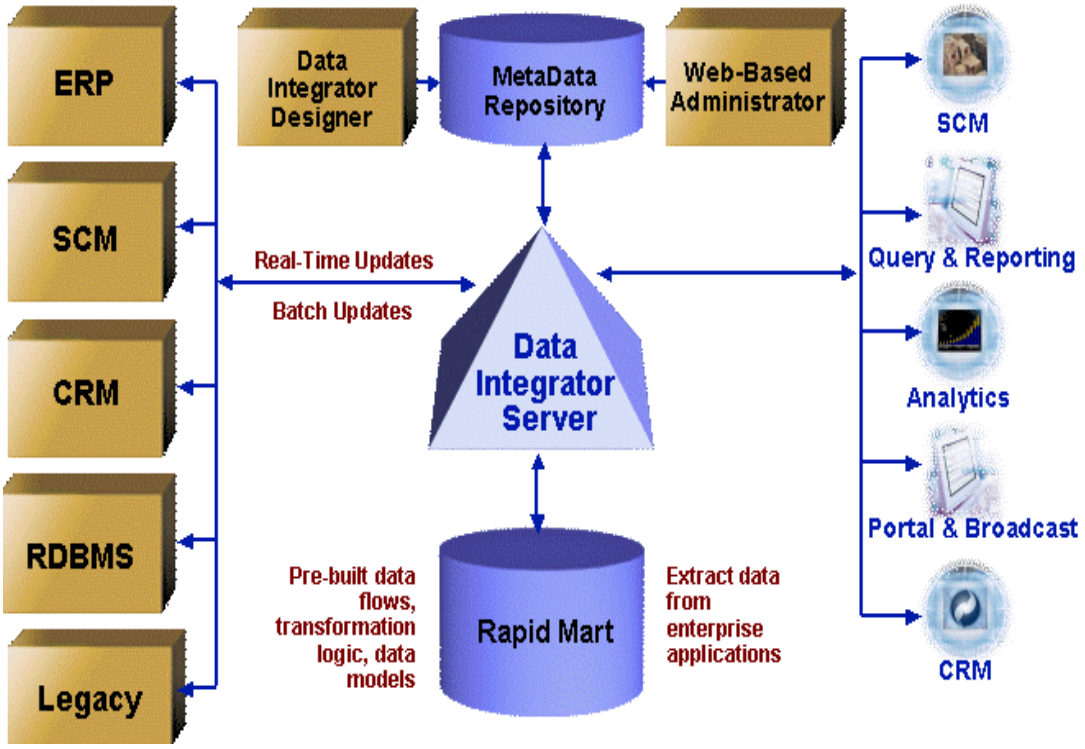
With the BusinessObjects Data Platform, enterprises can easily extract data in batch and real-time, and cache that data in relational databases. E-commerce and analytic applications can readily access this cached data. With this data:

- Entermprise can drive better relationships with customers, suppliers, and partners
- Enterprise decision makers can better understand and drive the business
- Enterprise can manage their corporate data as an independent asset
- With the Business Objects Data Platform, enterprises can use their data in such projects as:

- Sell-side and buy-side e-commerce, and participation in market places
- Data warehousing for analytics, reporting, and data mining
- Data migration for obsolescence, mergers, and acqusitions
- Master data management, such as a single view of the customer

The Business Objects Data Platform has two product lines:

- Data Integrator — The core technology
- Rapid Marts — Pre-packed information components that use the Data Integrator framework



Data Integrator

Data Integrator allows enterprises to build and execute applications that create and maintain business intelligence warehouses. Data Integrator RealTime module extends this functionality by including request-response processing logic needed to create and maintain databases that support e-commerce.

The Data Integrator data server allows enterprises to find, move, manipulate, and manage their information in both batch and real-time, ensuring that information is always up-to-date and semantically correct.

This Data Integrator architecture allows enterprises to intelligently combine real-time data movement with data staging. This architecture provides enterprises with the control required to maximize application performance, availability, and scalability.

Rapid Mart

A Rapid Mart is a prepackaged Data Integrator application that delivers data for a specific business purpose. Easily deployed and customized, a Rapid Mart includes the extraction, transformation, and data loading logic to support analysis or e-commerce. Business Objects offers a suite of fully-integrated Rapid Marts. For more information, see "What is a Rapid Mart?" on page 12

BusinessObjects products are available for a variety of platforms. With BusinessObjects products, an enterprise can access data across different packaged transactional applications, hardware, operating systems, and database management systems. Using the BusinessObjects Data Platform, an enterprise can deliver an integrated corporate-wide information architecture.

Benefits of the BusinessObjects Data Platform

The Business Objects Data Platform is the only integrated batch and real-time data platform. The Business Objects Data Platform provides enterprises the benefits of superior data management:

- Dramatically reduces implementation time for faster return on investment
- Delivers rapid response times for business decision makers, customers and partners
- Minimizes impacts on the back-office systems by intelligently caching and managing complex data access rules in batch and real-time
- Enables 24x7 data availability even when back-office systems are off-line
- Provides a single data environment for managing the diverse demands of information-rich interactions and transactional data flows
- Delivers pre-packaged Rapid Marts that can be easily customized to meet specific business needs

Enterprises can implement the Business Objects Data Platform in days, customize it within weeks, and rapidly deploy business analysis and e-commerce to increase the return on their back-office system investment.

Rapid Mart Architecture

chapter 3

What is a Rapid Mart?

BusinessObjects™ Rapid Marts provide packaged extraction of operational data to accelerate the deployment of business intelligence (BI). They are add-on products to Data Integrator™ that combine domain knowledge with data integration best practices to deliver pre-built data models, transformation logic, and data flows for extracting data from enterprise applications from SAP, PeopleSoft, Oracle, J. D. Edwards, and Siebel.

Rapid Marts deliver pre-packaged business content and data flows in a single data platform optimized for BI and analytic applications. Rapid Marts enable business users to report on and gain insight into their enterprise information. Rapid Marts allow you to extend your data platform to support business processes with trading partners, suppliers, and customers outside your organization.

BI tools and analytic tools can access Rapid Mart data directly through SQL queries or indirectly through a Data Integrator RealTime request-response system. You can implement Rapid Marts individually, or in any combination, to form a single platform that delivers the infrastructure for your company's internal and external information needs.

Because you can implement them quickly and customize them easily, Rapid Marts provide an immediate return on investment.

Analytic Rapid Marts

An Analytic Rapid Mart is an application that produces a predefined data warehouse for a specific area of business analysis. Analytic Rapid Marts are typically based on dimensional modelling. Examples of Analytic Rapid Marts include:

- Sales Rapid Mart
- Inventory Rapid Mart
- HR Rapid Mart
- Accounts Receivable Rapid Mart

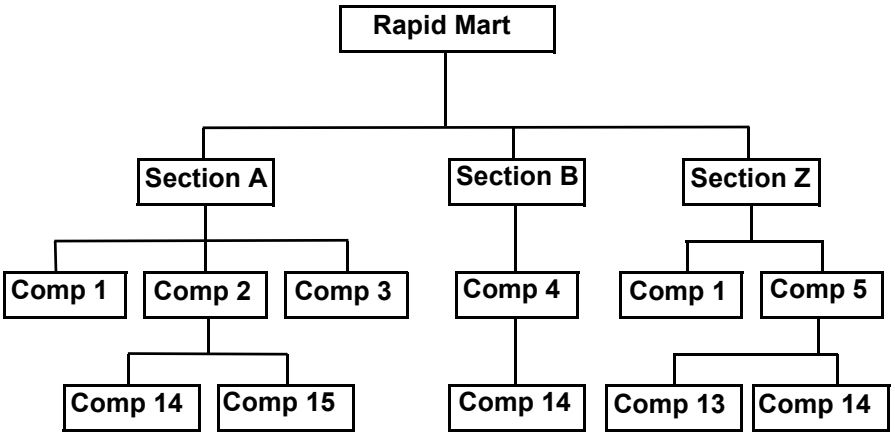
Components and Sections

A Rapid Mart is composed of multiple components. A component is a stand-alone work flow that completes a particular task, such as loading a specific dimension table. Components are work flows designed to be reused.

Components can contain other components. A section is a set of components that address a particular business problem or subject area, in most cases it is a complete Star Schema. A section is itself a component.

The same components can be used in multiple Rapid Marts. For example, the material component, which extracts information about materials bought, produced, and sold, is used in the Sales, Purchasing, Inventory, and Plant Maintenance Rapid Marts. Work flows that extract star schema “dimensions” are components. You can add a component to any Rapid Mart using a simple import procedure.

A Data Integrator job can include multiple instances of a component. For example, each section includes all the required dimension components. Therefore, a job with several sections may include several instances of a particular dimension component. Components are set to execute only once within a job. This “execute once” feature ensures that shared components do not cause duplicate data extraction. For more information about the “execute once” feature, see the Data Integrator Designer Guide.



Benefits of component architecture

The Rapid Mart component architecture lets you create a flexible solution that meets customers’ needs. For example, BusinessObjects Sales Rapid Mart includes a sales order section and a billing section (in addition to backlog and delivery sections). BusinessObjects Accounts Receivable Rapid Mart includes a customer items section (in addition to a customer totals section).

To meet the needs of companies who wanted to deliver order and invoice information over the Internet to their customers, Business Objects created a Customer Billing Rapid Mart. The Customer Billing Rapid Mart simply combines the sales order, billing, and customer items sections. The Rapid Mart component architecture made this a straightforward task.

The component architecture simplifies maintenance. With properly implemented components, you use the same data flow to load a particular table in all Rapid Marts with that table. This ensures that all Rapid Marts are consistent and use the same logic. When you fix an error or enhance a feature in a component, all Rapid Marts using that component benefit from the change.

Elements of a section

When you develop a section (the main component in a Rapid Mart), you must deliver several elements:

- Data model
- ATL file
- SQL DDL scripts
- DTDs
- Documentation
- Typical queries
- XML message samples
- Data for target database
- Universe/Reports samples

Data model

Each section requires a model that describes the target database for the business subject area or analytic question that the section addresses. The model is generally a star schema, with a fact table and associated dimensional table.

You can build a diagram of the model with a data modeling tool, such as ERwin or ER/Studio. This model helps others understand what data the section loads into tables and how the tables are related to each other. Many data modeling tools can generate a SQL data definition language (DDL) script that creates the target schema tables.

ATL file

Each section contains built-in extraction, transformation, and load logic. This logic is contained in Data Integrator work flows and data flows that load the target database for querying or that manipulate data and transact with a source in real-time. You create these work flows and data flows using the Data Integrator Designer.

Data Integrator stores the work flows and data flows, as well as the structure of the tables, files, and messages manipulated by the section, in its repository. You can export the contents of a repository or parts of a repository, such as a section, into an ATL file.

Exporting a section into an ATL file allows you to share that section. For example, you can import the ATL file containing that section into another repository that contains a different Rapid Mart. An ATL file is also the method for providing the end user with the Rapid Mart jobs on a CD.

SQL DDL scripts

Each section must have a set of SQL DDL scripts that create the required tables and indexes and a set of scripts that drop the tables and indexes. Customers use these scripts when installing the Rapid Mart. Customer DBAs may modify the scripts when customizing the Rapid Mart. You need to create DDL scripts for specific target database platforms.

BusinessObjects™ Rapid Mart products are developed on Oracle platform. An installation package for each Rapid Mart contains DDL scripts for DB2 and SQL Server platforms.

DTDs

A section in a Rapid Mart requires DTDs (document type definitions) for all the input output XML documents in the section's job, including real-time jobs. A DTD describes the elements (tags) in an XML document and the relationship among the elements. When an XML document describes a transaction, the DTD describes the data schema the transaction uses.

Documentation

Each section requires documentation that can be assembled into a User's Guide or other document that describes the Rapid Mart. There are several required parts of the documentation:

- Description

Provide a brief description of the section. In the description, explain what data the section contains and what type of analysis the section supports. Specifically, list the business function, type of analysis, and measures available in the section.

- Subject areas

  Provide a brief description of what the section does and appropriate information to assist those analyzing the data that the Rapid Mart extracts. For example, you might discuss how the back-end system processes data in this area, how the section processes the data, and how users might use the data.

- Reports

  Describe the types of analyses and reports users can produce using the section. Provide an example of a report that they can produce. List the joins they need within the section, and where possible, suggested joins with other sections.

- Using the section

  List any variables the section requires (the variable name, the format, and a description). Indicate whether variables are local to a particular section. Also list any special steps required when running the component.

- Technical implementation

  Provide technical details about the implementation of the section, such as describing the work flows that load data.

- Data schema

  List the tables that the section loads, including the fields, the primary keys, data types, and brief description.

Typical queries

Each section requires sample queries that show how customers can use or analyze data in the target database. Typically, these samples respond to the business questions that the section addresses. Include these sample queries in the documentation (illustrate the sample report with the SQL query and a screen shot of the results). The marketing and sales team can refine these samples for demonstration purposes.

XML message samples

Sections in Commerce Rapid Marts require samples of XML messages. Use these samples to test the real-time data flows in the Rapid Mart. The marketing and sales team can refine these samples for demonstration purposes.

Data for target database

To run the sample queries or test the sample messages, you must populate the section's target tables with some data. During development, you can extract data from a demo database, such as SAP IDES or Oracle Apps Vision.

Universe/Reports samples

Each section is included in a Universe including contexts, aliases, hierarchies, etc., some default users and groups with assigned security and reports

Data Integrator objects in a Rapid Mart

A Rapid Mart can contain several types of Data Integrator objects:

- Projects
- Jobs
- Work flows
- Scripts
- Data flows
- R/3 data flows
- Real-time data flows
- Custom ABAP programs

Projects

A Rapid Mart typically contains one or more Data Integrator projects. Each project contains one or more jobs, which can load data in batch mode or respond to real-time requests. For Analytic Rapid Marts, which usually only

contain batch loads, a single project is usually sufficient. However, for Commerce Rapid Marts, which usually contain real-time interactions, multiple projects are useful. Projects are a convenient way to group related real-time jobs.

Jobs

There are two types of jobs: batch and real-time. A batch job extracts, transforms, and loads data. A real-time job receives a request, extracts, transforms and loads data to support that request, and then responds to the request in real-time. The Data Integrator Designer can execute a job or the Data Integrator Access Server can execute a job as a request-response real-time service. A Rapid Mart groups jobs into projects.

Work flows

Work flows contain steps that define the order of job execution. Work flows call data flows, but cannot manipulate data themselves. You call work flows from inside other work flows or jobs. Components and sections are work flows. A Rapid Mart groups sections into jobs.

Scripts

A script is a collection of statements in Data Integrator scripting language. Typically, batch jobs start and end with scripts (an initialization and inalization script). You can also use scripts inside components to execute SQL statements, set variable values, or run Data Integrator functions.

Any variables used in a data flow must be initialized in a script. The script that initializes variables can be at the beginning of the job containing the data flow—the case in most Rapid Marts—or the script can be within the component itself.

Data flows

A data flow contains steps to extract data from one or more sources, transform the data, and load the data into one or more target tables.

R/3 data flows

Data flows that extract data from SAP R/3 are called R/3 data flows. Data Integrator translates steps you define in an R/3 data flow into ABAP and then passes the ABAP program back to your SAP R/3 system for execution. The resulting table or file resides on the SAP R/3 system to be used as a source in the parent data flow.

R/3 data flows

Data flows that extract data from SAP R/3 are called R/3 data flows. Data Integrator translates steps you define in an R/3 data flow into ABAP and then passes the ABAP program back to your SAP R/3 system for execution. The resulting table or file resides on the SAP R/3 system to be used as a source in the parent data flow.

Real-time data flows

Data flows that respond to requests in real-time are called real-time data flows. A real-time data flow performs predefined extraction, transformation, and load operations, and responds appropriately to the message content. You design real-time data flows in the Data Integrator Designer. The Data Integrator Access Server is the process that starts real-time data flows. You manage and monitor real-time data flows using the Data Integrator Administrator.

Custom ABAP programs

In some cases, the ABAP programs that Data Integrator generates automatically will not meet your requirements. This is the case, for example, if you want to extract data from certain types of cluster tables. In these cases, you can create custom ABAP programs and embed them as transfoms.

Development Process

chapter 4

Determining requirements

Before developing a Rapid Mart section, you need to know the business requirements that you must satisfy. Rapid Mart sections are developed to meet a market need. Typically, Business Objects partners and systems integrators receive requirements for a specific solution from their customers.

A business requirements document describes these requirements—the basic needs and subject areas that the section must address. Ideally, the business requirements document contains sample reports or report layouts. Sample reports are a good method for communicating requirements. The document also describes the scope of the section, clearly listing inclusions and exclusions.

Use the business requirements document as the starting point for designing and developing the Rapid Mart section. During construction, you can refine the requirements document to reflect changes in scope, requirements, and other realities.

Designing the data model

After the requirements are defined, you need to create a model of the target database that the Rapid Mart section will load. To create this data model, you need sufficient knowledge of the subject area that the section addresses.

A data model identifies all the required tables (both fact and dimension), columns, indexes, and their attributes (such as primary keys and comments or descriptions). The model also shows the relationship between different tables. The model lays the foundation for the physical design of the database.

You can create a data model with a graphical data modelling tool, such as Platinum's ERWin or Embarcadero's ER/Studio. You can also design and create a target table using the Data Integrator Designer. In the Designer, create a data flow and add a template table as the target. After you have designed the table, you can right-click on the template table and select **Make Table**. Data Integrator creates a physical table in the target database, with the specified primary key. The data modeling tool can detect and import this physical table.

The documentation for a section includes output from the modeling process. See "Documentation" on page 15 for more information.

Normalization and denormalization

You want sections to provide data that is easy to query. At the same time, you do not want to compromise the data integrity, redundancy, and storage requirements. Use judgment to strike a balance between usability and normalization.

For Analytic Rapid Marts, a section should be as close to a dimensional star (or snowflake) schema as possible. If your design includes more than one fact table, review the section's design and consider splitting the section into more than one section.

When solving a specific customer's problem, design the data schema in a format that is compatible with the customer's business intelligence tool—the tool the customer will use to report on the target database.

Primary, foreign and surrogate keys

Every table in the target database should have a primary key. When beneficial, use a surrogate key as a primary key, the natural primary key might be still needed for delta loading. Often, surrogate keys are an easy way to address the history preservation of changing dimensions. When using surrogate keys, it is often a good idea to carry the most-used natural key columns into related tables, also.

For more information about surrogate keys, see"Using surrogate keys" on page 71.

Views

Often, an easy-to-load structure is not easy-to-query. You can address this problem with views. With Data Integrator, you can often improve performance by creating a view and loading data from that view rather than using the underlying tables directly.

DTDs

A DTD is the model for an XML message. DTDs must be designed to capture and address the business problems that the Rapid Mart covers.

Nest elements when there are master-detail relationships (such as order header and line item), and when it makes sense to group related fields. Be careful, however, not to overuse nesting. When overused, nesting can become cumbersome to implement and maintain.

Initiating a project

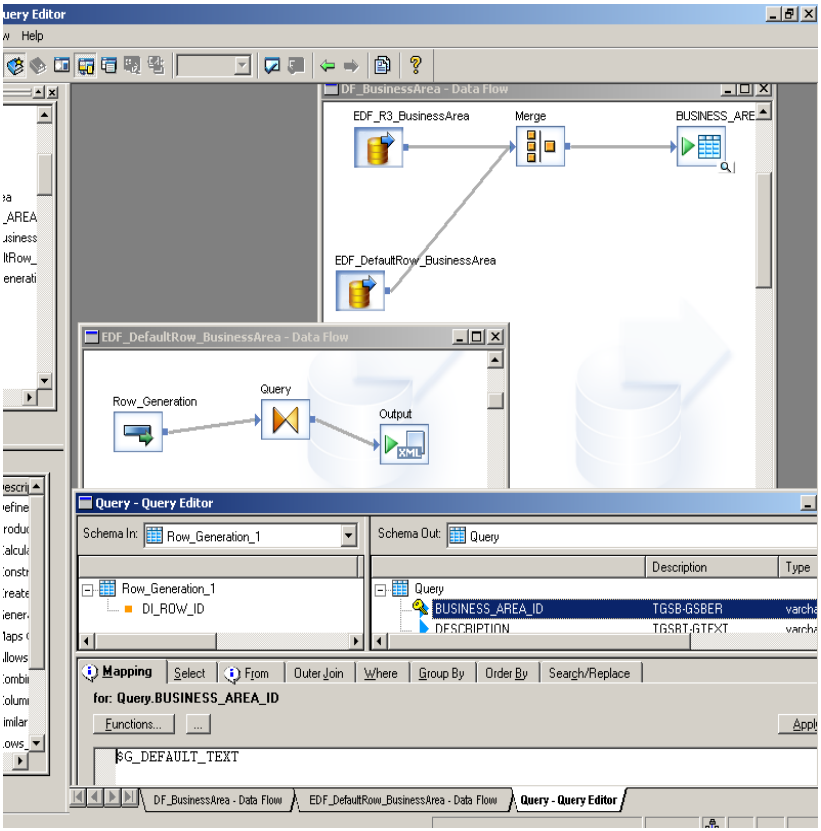
When initiating a project to develop a Rapid Mart section, you must complete several steps:

- Determine the project name-both a long name and an abbreviation, such as “Sales” and “SA”.
- Set up external systems, such as a version control system (for accompanying documents) and bug reporting system, to support the project. Use the Data Integrator Designer to initiate the development environment:
  - Create a new Data Integrator repository, including datastores for source and target systems
  - Create a new target database
  - Import any components from an existing Rapid Mart that you are going to use or modify in the new section
  - Adjust your environment properties
    - Set your R/3 datastore properties
    - Set your target datastore properties
    - Check that all drives are mapped to the appropriate directories
  - Import the tables you will be using for sources and targets

Creating new data flows

After you design your data model and initiate the project, you are ready to create the data flows that load the target tables. To create the data flows, you must know the source column mappings and transformation logic for each target column. The Data Integrator Designer shows the metadata of source tables in a graphical format. In the Data Integrator query transform, drag and drop columns from the source to the target, and then make any naming, mapping, or transformation modifications. Data Integrator provides metadata reports that list mappings and transformations. You can use these reports for your own documentation.

Occasionally, there are multiple ways to extract the same set of data. Choose the method that offers the best balance between load and query performance, and that is easy to maintain and support in the long-term. When performance is not an issue (for example, if the data set is small), use the method that provides better readability and that is easier to maintain and support.



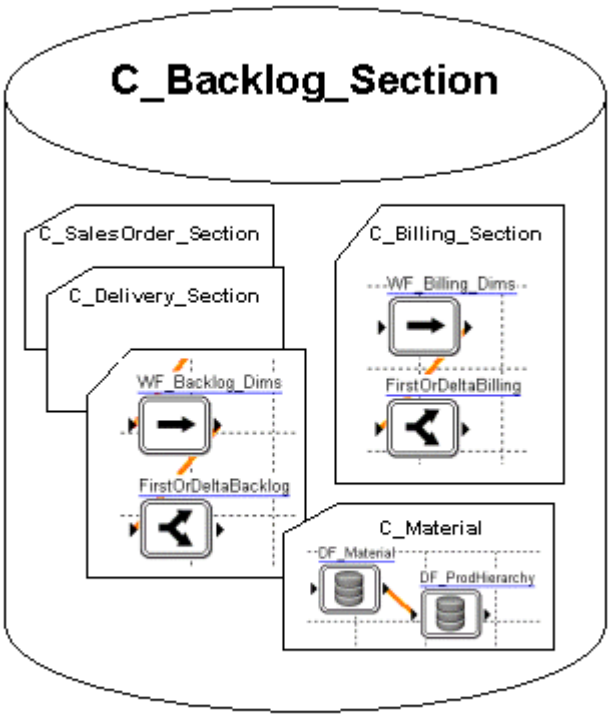
The extraction and common transformation logic is capsuled in an embedded dataflow. Therefore the same logic can be used in various places. Same Extractor might be used in First and Delta logic data flow, or the extractor is reused for another rapid mart version, e.g. loading not Oracle but SQL Server or even SAP BW.

Further more, to ensure foreign keys are valid it might be necessary to create one (or many) additional rows in dimensions tables to which fact tables rows will refer to in case no valid information is available for that column. In that case an additional Embedded Dataflow creating the rows from either a Row_Generation transform (one row) or an source table (multiple rows).

Creating components

After you create the data flows that load the target database, you need to organize those data flows into work flows and components, and ultimately into a Rapid Mart section.

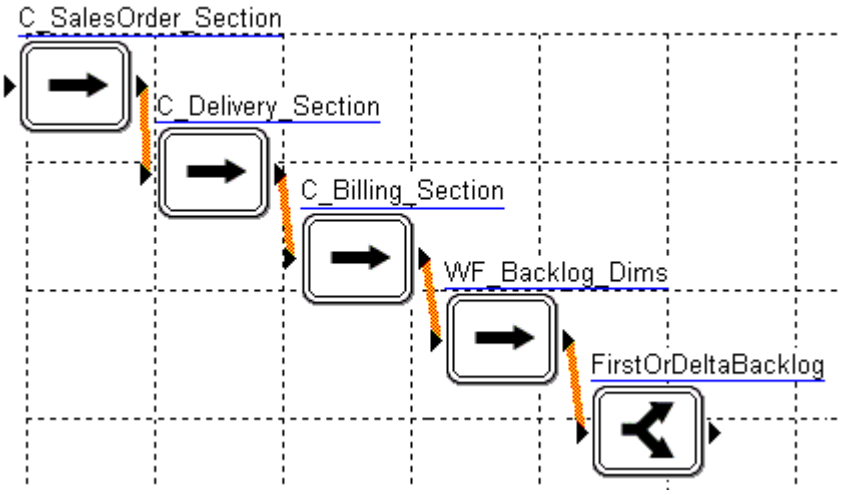
Rapid Mart section



Anatomy of a Component

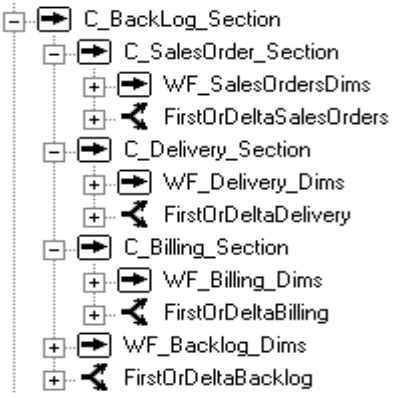
A Rapid Mart section can include other Rapid Mart sections (for example, sections that load related star schemas), components that load an individual table or that load a set of tables, and “regular” work flows that contain a unique grouping of components for this Rapid Mart. Regular work flows-work flows that are not components—are unique to a Rapid Mart and are not intended for reuse.

For example, the Rapid Mart section, C_Backlog_Section, consists of three other sections, a work flow, and a conditional object.



C_SalesOrder_Section, C_Delivery_Section, and C_Billing_Section are Rapid Mart sections—each loads a complete star schema of fact and dimension tables for a particular area of analysis (billing, delivery, and sales orders). WF_Backlog_Dims is a work flow that contains components that load the dimension tables for the backlog section. FirstOrDeltaBacklog is a conditional object that loads only one fact table.

The following diagram shows another view of the backlog section—the expanded project area in the Data Integrator Designer:



Note that the sales order, delivery, and billing sections each include a work flow that loads the section's dimensions and a Data Integrator conditional object (entitled "FirstOrDeltaSalesOrders", "FirstOrDeltaDelivery", and "FirstOrDeltaBilling") that loads the section's fact table. The work flow WF_Backlog_Dims contains several components that load the dimension tables.

Dimension table components

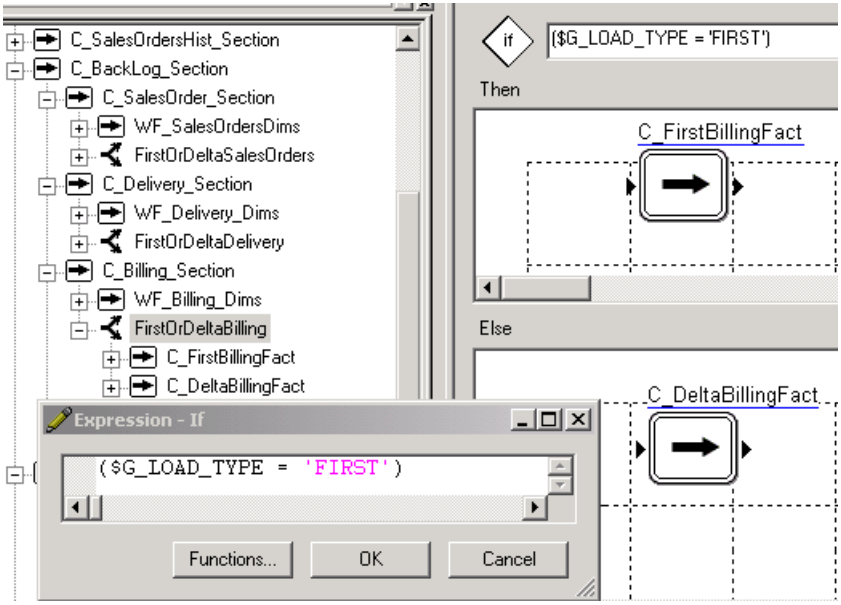
The simplest component loads a single dimension table. Components that load dimension tables typically contain conditionals that determine when to load the dimension table.

For example, the work flow WF_Billing_Dims loads the dimension tables in the billing section. This work flow includes the component C_Currency.

For those dimension tables that do have change data capture logic, the procedure is the same as described in the following section.

Fact table components

Components that load fact tables also use conditionals to specify different load logic for initial and incremental loads. During incremental loads, change data capture logic is often implemented.

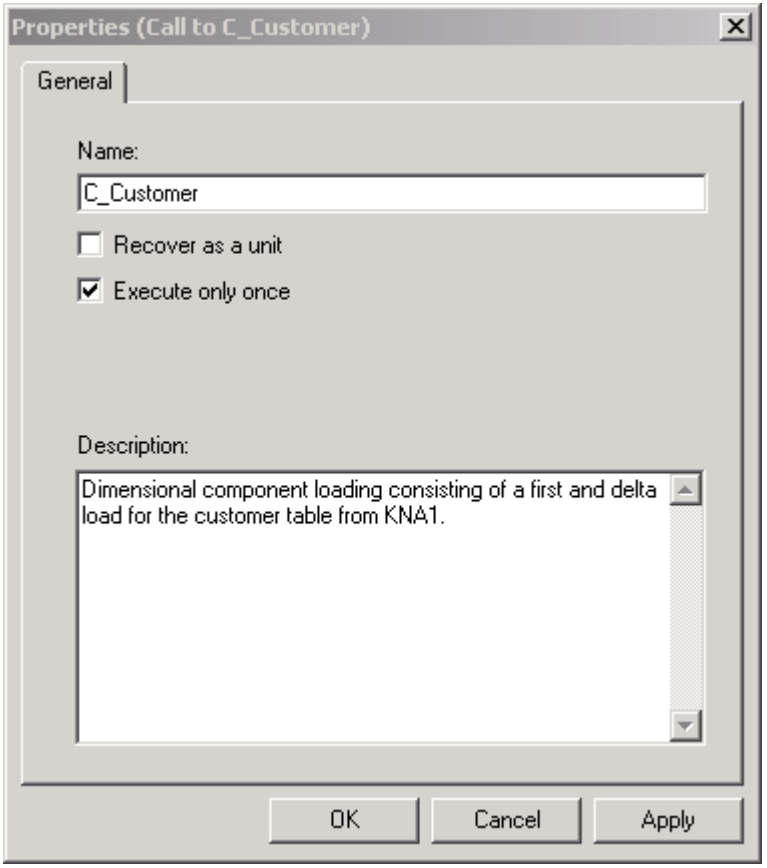


For example, the billing section (C_Billing_Section) loads the BILLING target table. The conditional determines whether the job is an initial load ($G_LOAD_TYPE='FIRST'). If so, the job executes the **Then** block.If not, 'DELTA' is assumed, and the job executes the **Else** block.

Note that the **Then** block contains a work flow and the **Else** block contains another work flow. This is because the initial and incremental loads logic requires several data flows to load the target fact table.

Executing identical data flows

Several sections can call the same components (see "Components and Sections" on page 12. Therefore, a job might call a single data flow or work flow more than once. For example, C_Customer (a dimension table component that loads a table called CUSTOMER) is called by both the billing section and the sales order section. To avoid re-loading the same table, set each component to execute only once.: select the Execute only once check box on the work flow's Properties window.



Select the **Recover as a unit** check box when sub-components of a component should be re-run if a failure occurs in the middle of execution, e.g. a script deleting data that will be loaded in the DF build a unit

Creating Data Integrator jobs and projects

A Rapid Mart typically contains one or more Data Integrator projects. Each project contains one or more jobs, which can perform either batch loads or real-time operations. For Analytic Rapid Marts, which usually only contain batch loads, a single project is usually sufficient. However, for Commerce Rapid Marts, which usually contain real-time interactions, multiple projects are useful. Projects are a convenient way to group related real-time jobs.

A job that performs batch loads executes components to load the Rapid Mart tables. This job usually contains:

- An initialization script which completes several steps:
  - Sets parameters for the job to use
  - Initiates the execution status framework
- Sections that load a set of tables. A section, in turn, consists of:
  - Other sections that this section requires.
  - A work flow that loads the reporting dimensions applicable to the section.
  - Work flows that load holding tables, intermediate tables, and auxiliary tables that support the main tables. Some of these work flows may be reusable components, which may be incorporated into other components or Rapid Marts. Components also have the run once feature enabled.
  - A conditional object that determines whether the job executes the initial or incremental load logic.
  - Work flows that load the main tables contained in the subject area. When a Rapid Mart is designed based on dimensional modeling, these tables are called fact tables. A section usually loads a single main table and any additional tables that depend on the main table or strongly relate to the main table. For example, there may be a section that loads a table containing individual purchase orders and a purchase order summary table. Together, these two tables make up the subject area of purchase order analysis.
- A finalization script that updates the job execution status table.

Components in a Rapid Mart are re-usable. For example, the vendor component (C_Vendor), which contains information about vendors, is used in the Inventory and the Purchasing Rapid Marts.

Use a simple export-import procedure to add an existing component to any Rapid Mart. After you import a component, you can add it to your existing Data Integrator objects (jobs, work flows, and so forth), without worrying about using that component more than once inside a Data Integrator job. Components are set up to execute only once within a job. This run once feature ensures that sharing components across Rapid Marts does not result in unnecessary duplication of data extraction from SAP R/3.

In the initialization script, you must set the initial values for all the variables required by the imported components. If you import a component that uses a variable, such as $CURRENT_DATE, you must initialize that variable to a specific value before you can execute the job.

- To initialize variables for a combination of components
  1. Create variables both local and global variable at the job level.Include the prefix $G_ in the name of global variables.
  2. Edit the initialization script (typically called InitScript), which is the first object a job executes, to set the initial values of the variables you created.

     ```
     #
     # Initialize variables:
     #
     $G_SDATE = to_date(sysdate(), 'DD-MM-YYYY');
     $CURRENT_DATE = sysdate();
     ```
  3. For local variables, create parameters to pass the value of the variables to any components that require them. Give parameters the same name as the corresponding variable.
  4. Use the Variables and Parameters window in Data Integrator to pass the values of the local variables from the job to the required component.

Designing for recovery

When designing a Rapid Mart, you must be sure that you can recover from failed executions. This section discusses two techniques that you can use to ensure that you can recover and provides more details about using the Rapid Mart recovery framework, which you can build into a new Rapid Mart.

Recovery framework

When executing a Rapid Mart, it is important that you do not load any duplicate data and that you do not extract data redundantly, even in cases of failure. There are two mechanisms to ensure that you do not load duplicate data.

- Data Integrator automatic recovery feature
- Rapid Mart recovery framework

Data Integrator automatic recovery feature

With automatic recovery, Data Integrator records the result of each successfully completed step in a job. If a job fails, you can choose to run the job again in recovery mode. During recovery mode, Data Integrator retrieves the results for successfully completed steps and reruns uncompleted or failed

steps under the same conditions as the original job. For recovery purposes, Data Integrator considers steps that raise exceptions as failed steps, even if the step is caught in a try/catch block.

To use the automatic recovery feature, you must enable the feature during the initial execution of a job. In the Execution Properties window, select the **Enable Recovery** check box.

The automatic recovery option is useful during testing and development phases. However, use caution with this feature during a production run because the data may lose its integrity if significant changes in SAP R/3 source tables occur between the failed run and the recovery run.

For example, suppose two tables contain related data but are loaded in separate work flows. In this case, the job can fail after successfully completing the work flow that loads the first table but before completing the work flow that loads the second table. During recovery, Data Integrator does not re-run a successfully completed work flow; therefore, the first table is not loaded again. However, Data Integrator will run the work flow that loads the second table during the recovery job. If the work flow loads the entire table regardless of dates, the second table will contain data added to SAP R/3 between the initial run and the recovery run, but the first table will not contain this data. As a result, the two tables will be inconsistent.

Rapid Mart recovery framework

A Rapid Mart contains an external execution status table (AW_JOBEXECUTION) that tracks the status of a job's execution and the time data was extracted. By using the execution status table, a Rapid Mart ensures that data for the proper dates is extracted from the source datastore.

With each execution, a Rapid Mart job extracts data between a start date and an end date. If the job fails and is rerun on the same day or on a subsequent day, the new job uses the failed job's start date to ensure that all the data is extracted contiguously.

The recovery job first checks the status of the last execution of the job. If it failed, the recovery job determines the ending date of the last successful execution. This date becomes the starting date of the next execution of the job. All work flows are re-executed using the same starting date to ensure consistency. For example, if a failed job is restarted on the same day (without using the automatic recovery feature), the restarted job refreshes all tables because the last execution of the job on that day was not successful.

If a job continues to fail over several days, the job would continue using the same start date used on the first failed attempt. This ensures that when the job is eventually successful, it extracts the complete set of data that changed since the start date.

To maintain consistency with data flowing to SAP R/3, a Rapid Mart always overlaps one day's worth of data in incremental loads. For example, if the end date is June 3, the incremental load job extracts data entered or changed on June 3. The next incremental load job extracts data starting with the June 3 date.

Up to one day's worth of duplicate data may be extracted each time the job runs; however, work flows are designed to detect and delete duplicate data from the target data table before reloading the refreshed data. To do this, many work flows use the auto correct load option. The auto correct load option checks for an existing row with the same key as each row flagged as an insert or update activity. If the row exists, this option updates the row. If the row does not exist, this option inserts the row.

Using the Rapid Mart recovery framework

The Rapid Mart recovery framework uses two mechanisms to check the job execution status and control execution:

- An execution status table that stores one record for each job. That record contains:
  - Status of previous job execution
  - Starting and ending times of the last attempted execution of the job
- An application programming interface (API) that updates the status table and supports the implementation of restartable jobs and flows

The execution status table

The Rapid Mart installation procedure creates a table named AW_JOBEXECUTION in the target database.

The execution status table has four columns

Table column	Description
NAME	The name of the job being executed. If the same target datastore is used for more than one repository, you must ensure that all job names are unique.
STATUS	The status of this job after the last execution. Possible status values are: started, done, or none.
EXTRACTLOW	Last attempted start date of data retrieval. This date is also the ending date of the last successful run of the job..
EXTRACTHIGH	Last attempted end date of data retrieval

How job status is maintained

A Rapid Mart records a job status for each invocation of a particular job. There are three possible values for job status:

- None - The job status is none when the job has never been executed.
- Started - Job status is set to started when a new invocation of a job starts.
- Done - Job status is set to done when a job completes successfully.

When a job is started, a Rapid Mart checks the job's existing status. If the previous execution of the job did not complete successfully that is, the job status is not done a Rapid Mart resets the starting date for data extraction ($G_SDATE) from the value specified in the job initialization script to the ending date of the last successful run (EXTRACTHIGH). You can override the reassignment of the starting date by forcing the starting date to be reset. See "Executing a job with the reset option" on page 35. A Rapid Mart never changes the ending value ($G_EDATE) from the value specified in the initialization script.

After checking the job's existing status, a Rapid Mart sets the status of the job to started, and starts a new execution of the job.

When a job successfully completes, a Rapid Mart sets the job's status to done. If the job does not complete successfully, the job's status remains set to started. The EXTRACTLOW and EXTRACTHIGH dates remain set to the current values.

The execution status API

The execution status API updates the execution status table to control gaps in the time sequence of data extraction.

The API contains two external functions. These functions mark the start and termination of jobs.

```
* AW_StartJob ($jobname input, $run_mode input,
   $load_type input, extractlow input/output,
   extracthigh date input/output)
```

The InitializeJob script calls this function to initialize a job and check the last execution status. When called for a new job execution, the function inserts a new row into table AW_JOBEXECUTION. When called for a restarted job execution, the function:

- Checks the run mode to see if you set the reset option.

  If $RUN_MODE is RESET, the function resets EXTRACTLOW and EXTRACTHIGH to the $G_SDATE and $G_EDATE values specified in the initialization script.

- Checks the last execution status of the job.

  If the job status is *done*, the function sets EXTRACTLOW to the previous value of EXTRACTHIGH, the last successful end date; next, the function sets EXTRACTHIGH to the value of $EDATE, which you set in the initialization script. The function returns the values in EXTRACTLOW and EXTRACTHIGH and the Rapid Mart uses these values to update $G_SDATE and $G_EDATE.

  If the job status is *started* , the function does not change the value of EXTRACTLOW. Instead, this value remains the end date of the last successful execution. The function does change EXTRACTHIGH to the new value set in $G_EDATE. The function returns the EXTRACTLOW and EXTRACTHIGH values, and the Rapid Mart uses these values to update $G_SDATE and $G_EDATE.

*AW_EndJob(job name)

The EndJob script calls this function to change the job status value in the AW_JOBEXECUTION table to *done* when the job executes successfully.

Executing a job with the reset option

You can use the reset mode to force a Rapid Mart to assign a new starting extract time for data extracts. Use reset mode to force the starting and ending dates to remain the same as the values set in the initialization script.

Note: Using this option may cause gaps in the time sequence os date loaded into target tables.

To specify reset mode, edit the job's initialization script and uncomment the line that sets $RUN_MODE to RESET. When using reset mode, be sure that the variable $G_SDATE is set to the appropriate starting date so that no gaps occur.

Programming Conventions

chapter 5

Version Number

A Rapid Mart version number indicates a specific Rapid Mart installation. Following a typical software engineering convention, the Rapid Mart development team codes each Rapid Mart with four numbers, separated by dots:

- Major feature
- Minor feature
- Maintenance release
- Emergency bug fix

For example, a release of BusinessObjects Sales Rapid Mart has version number SA 3.1.0.0

The Rapid Mart version is stored in a work flow named WF_VersionID_<*Rapid Mart Abbreviation*> , such as WF_VersionID_SA. This work flow helps Rapid Mart developers determine the specific version deployed at a certain site.

The batch job that loads the Rapid Mart calls this work flow after completing the initialization script. Locating the work flow after the initialization script ensures that Data Integrator prints the name and version of the Rapid Mart in the trace file for each run.

For example, this is the code for this work flow in the Customer Order Rapid Mart.

```
CREATE PLAN WF_VersionId ( )
BEGIN
BEGIN_SCRIPT
#Copyright 2002 Business Objects Data Integration, Inc. All
   rights
reserved.
print('Customer Order Rapid Mart 2.0.0.0');
END
END
```

Use these rules to determine when to change a digit in the Rapid Mart version number:

- A major release is expected to have significant changes.
- A minor release is expected to have additional functionality, but not significant amounts.
- A maintenance release has no additional features, only bug fixes.

- Emergency bug fixes (EBFs) are increased by 1 whenever a release includes one or more bug fixes or improvements. If a bug is fixed and released, and later a regression is encountered, a new EBF is generated and the release number appropriately increased.
- Periodically, the latest version of a Rapid Mart and its associated EBFs might be promoted to a new maintenance release or even a new minor or major feature release.
- When increasing a number in the release hierarchy, all lower-level numbers are reset to zero. For example, if the minor feature number changes from 1 to 2, then the maintenance release and emergency bug fix numbers are reset to 0. That is, SA 3.1.2.4 would change to SA 3.2.0.0.

Naming guidelines

Guidelines for names vary with the type of object you are naming. This section provides general naming guidelines for different types of objects:

- Data Integrator objects
- Abbreviations
- User names
- File names

Data Integrator objects

For first-class objects, such as jobs, work flows, data flows, or R/3 data flows, use a three-part name:

```
<object type>_<object name>_<source ERP>
```

Possible object types include:

- DF data flow
- EDF --- Embedded Dataflow for extraction
- EDF_DefaultRow --- Embedded Dataflow creating the rows needed to valid all FK relationships
- RTDF real-time data flow
- WF work flow
- C component
- R3 R/3 data flow

Possible source ERPs include:

- SEA Siebel Applications
- ORA Oracle Applications
- SAP SAP R/3 system
- PSFT PeopleSoft
- JDE JDE OneWorld
- RTK Retek

Placing the object type in the name allows you to identify the type of object from the project tree or the object library. Do not use underscores in the object name. For example, DF_Delivery_Fact is wrong; DF_DeliveryFact is correct.

Clearly mark objects involved in the incremental loads by using the prefix “Delta” in the object name, such as DF_DeltaMaterial.

For second-class objects, such as scripts or conditionals, create a name that indicates the object’s function, such as ReloadProduct or InitScript.

Do not use the words Load or Get in any object name. Data Integrator always gets data from a data source and loads the data into a target database. Similarly, never use the work Check in the name of a conditional.

For more detailed information about naming conventions for specific Data Integrator objects, see “Naming Data Integrator Objects” on page 43

Abbreviations

- BusinessObjects Development maintains a list of abbreviations used in BusinessObjects products. BusinessObjects developers are required to consult this list when using abbreviations. Business Objects encourages partners to obtain and use this list. This list contains standard abbreviations of common business and technical terms. Use these abbreviations as a guide throughout Rapid Mart development, such as when naming objects or entering descriptions.

User names

Business Objects Development uses RM_USER (uppercase) as the name of the table owner. A common table owner name enables BusinessObjects to share components easily and allows components to coexist in the same Data Integrator repository.

File names

Rapid Mart components typically contain several types of files, each with separate naming conventions. This section discusses these file types:

- SQL script files
- Data model file
- ATL file
- DTD and XML files

SQL script files

For a particular database, a Rapid Mart requires five SQL DDL scripts:

- Create table script
- Create table with comments script
- Create index script
- Drop table script
- Drop index script

The convention for naming these script files is:

```
<Rapid Mart Abbreviation>_<Functional Name>_<Database
    Name>_<Schema Type>.sql
```

where:

- *Functional Name* is:
  - Create
  - Drop
- *Database Name* is:
  - Ora
  - Ifmx
  - MSSQL
  - DB2
- *Schema Type* is:
  - Tables
  - Tables_Comments
  - Indexes

Note: If you are creating a component rather than a Rapid Mart, you can use the component name in place of the Rapid Mart abbreviation.

Data model file

The convention for naming the data model file is:

```
<Rapid Mart Abbreviation>_Model_<Database Name>.ER1
```

ER1 is the extension for an ERWin file. When using other data modeling tools, use the appropriate extension for that application.

ATL file

The convention for naming the ATL file is:

```
<Rapid Mart Name>_Rapid_Mart.atl or
<Rapid Mart Name>_<Section Name>.atl
```

DTD and XML files

The convention for naming DTD and XML files is:

```
<Service>_<Direction>.<Format>
```

where:

- *Service* is a description of the function
- *Direction* is:
  - in for a request
  - out for a response
- *Format* is the file format, such as XML or DTD

Some examples are:

- OrderCreate_in.DTD
- OrderCreate_in.XML
- OrderCreate_out.XML

Example:

The Profitability Rapid Mart contains four files:

- PA_Create_Ora_Tables.sql - Profitability create table script in Oracle database
- PA_Drop_Ora_Indexes.sql - Profitability drop index script in Oracle database
- PA_Model_Ora.ER1 - Profitability ERWin file
- Profitability_Rapid_Mart.atl - Profitability ATL file.

Naming Data Integrator Objects

This section describes conventions for naming specific Data Integrator onjects.

Projects

Each Rapid Mart has one or more projects. Name batch projects either *<Rapid Mart Name>_Rapid_Mart* or *<Rapid Mart Name>_Batch*. Name real-time projects *<Rapid Mart Name>_RealTime* with a suffix for the group name, if convenient.

For example, project names include:

- Inventory_Rapid_Mart
- Sales_Rapid_Mart
- Customer_Order_Batch
- Customer_Order_RealTime
- Customer_Order_Config_RealTime

When real-time jobs become obsolete, store them for backward compatibility purposes. Specifically, place them in a project that contains a suffix indicating the original version, such as Customer_Order_RealTime_2.0.0.

Jobs

Name batch jobs *<Rapid Mart Abbreviation>_Load*. This is the only case where you include the word “load”.

Name real-time jobs to match the real-time data flow they execute, replacing the “RTDF” prefix with “Job”.

SAP R/3 job names

The job name in SAP R/3 should match the name of the R/3 data flow. This job name *must* be less than 25 characters; longer names are unacceptable to SAP R/3.

Work flows

Typically, work flows group a set of data flows that complete a related task. A work flow’s name should indicate the types of data flows that the work flow groups. For example:

- WF_SalesDimensions - Loads dimension tables for sales, such as sales organization, distribution channel, division, and so forth.
- WF_CompanyStructureDimensions - Loads dimension tables with information about a company's structure, such as company code, controlling area, and so forth.
- WF_PricingData - Loads pricing tables used to calculate price.
- WF_VendorDimensions - Loads dimension tables with information about vendors (assuming you have at least two vendor-related tables).

When a work flow loads a single table or file, the table name or file name should appear in the work flow name, such as WF_Customer.

Components

The naming convention for components is as follows:

- All components begin with the letter "C" for "component" followed by an underscore
- Rapid Mart sections end with the suffix "_Section"
- Components that load a table begin with "C_" followed by the name of the table (or main table) loaded

Note: BusinessObjects recommends that non-BusinessObjects developers (customers or partners) add a two- or three-letter prefix to the names of components, work flows, and data flows, such as USR_C_Customer. Aprefix identifies which objects are part of a BusinessObjects product and which were created on-site.

Variables and parameters

Variable names must indicate what they represent. All variables start with a $ sign. Usually, variables that users can set are named in uppercase letters. All parameters used to pass the values of variables should have exactly the same name as the variable. Some examples are:

- $FISCALYEAR
- $RUNSTATUS
- $VALID_DATE

Whenever possible, consider using global variables to avoid passing parameters down a chain that calls multiple work flows and data flows. Use a "G_" prefix in the names of global variables, such as $G_SDATE.

Standard variables

There are some standard variables that all components in a Rapid Mart use. Declare these standard variables globally. These variables include:

- $G_SDATE
- $G_EDATE
- $G_LOAD_DATE
- $G_LOAD_TIME
- $G_LOAD_TYPE
- $G_LANGUAGE
- $G_REBUILD_INDEXES = 'N'
- $G_DEFAULT_TEXT = '?'
- $G_DEFAULT_DATE = 1900.01.01
- $G_DEFAULT_NUMBER = -1

The values for the global variables are set via the job properties to defaults, excluding EDATE, LOAD_DATE and LOAD_TIME. The values for these are either set on job execution or the initalize script will check if they are null and set them appropriately.

Data flows

Data flow names must indicate the tables that the data flow loads. Use discretion when naming "helper" data flows. Sometimes, a work flow must stage data, using a helper data flow, and load the final target table in multiple passes. When naming the helper data flow, use a modifier before or after the table name.

For example, the name of the data flow that loads the Currency table is DF_Currency.

If the data flow uses staging files or tables, the data flow's name includes the target table with a prefix that indicates the reason for the staging table. For example, if the work flow retrieves currency data from many sources, the data flow that merges that data into a final target table might be called DF_MergeCurrency.

Embedded Dataflows are named with a prefix 'EDF'. An example is given below:

EDF_R3_Currency_SAP

R3 - area of extract

An Integration Rapid Mart might extract from two different source systems, this is the indicator of the type of Extractor.

In case the Extractor is generating the default rows to enable all foreign key relationships to be valid the source system is called "DefaultRow" like EDF_DefaultRow_Currency_SAP

Query transforms

Assign meaningful names to queries if more than one query is in a data flow, or if the query is doing something that you want to recall. For example, possible query names are BAPIQuery, CalcPeriods, and RemoveDups.

R/3 data flows

Like regular data flows, the name of an R/3 data flow should indicate the table that the data flow loads. Helper data flows or intermediate data flows can have modifiers. For example, the name of the R/3 data flow that loads the Currency table is R3_Currency.

Note: The R/3 data flow name is initially used as the R/3 job name. Therefore, you must keep the name length less than 25 characters to avoid creating an invalid R/3 job name.

ABAP programs

You can set the name of the ABAP program name that an R/3 data flow produces. BusinessObjects names these ABAP programs ZAW4<SAP *Table Name*>.

This convention ensures that the program name is unique amongst all Rapid Marts and components that Business Objects ships. During installation, if you choose not to adhere to this convention, you can replace *ZAW4* with *ZAWX* where x is a unique identifier across products.

<SAP *Table Name*> is the name of the driving table. If you use the same table twice, alter the name slightly.

Transport file names

Within the R/3 object, the name of the transport files should be the name of the driving table in lower case with a ".dat" suffix, such as currency.dat. If you are staging data or doing an incremental load, or the R/3 data flow name does not match the table name for some other reason, then the transport file

should match the R/3 data flow name without the R/3 prefix. The transport file name should not exceed 32 characters, which is the limit in a generated ABAP program.

Real-time data flows

Use the prefix RTDF_ in the names of real-time data flows. For example, a real-time data flow that creates an order is named RTDF_OrderCreate.

Datastores

Always add the "_DS" suffix to datastore names. Create the source ERP datastore using the abbreviation for the source ERP system, such as R3_DS, PSFT_DS, SIEB_DS and ORA_DS.

Always create the target datastore with the name "RM_DS". If you need any other datastores, name them in a way that indicates the datastore's purpose. For example, "STAG_DS" could be an intermediate datastore to stage data.

Tables and indexes

Table names should be at most 18 characters in length, to support certain RDBMSs. Use the abbreviation guidelines to name tables (see "Abbreviations" on page 40 for more information). Indexes should start with X and be at most 8 characters long.

Use upper case letters for all table and index names in the physical model in ERWin (or other modelling tools). You can use underscores ("_") to separate words, since mixed cases are not usually preserved.

Table types

There are five types of Rapid Mart tables:

- Application
- Staging
- Holding
- Auxiliary
- Lookup

Each table type contains specific characteristics.

	Application	Staging	Holding	Auxiliary	Lookup
Description:	These are the final tables that compose the Rapid Mart data model	These are temporary tables - working areas that can be destroyed after a job runs.	These are intermediate tables that survive from run to run.	These are other tables that do not fit in other categories	These are used as source tables in case there is no such information in the source ERP system
Naming convention:	If it is a fact table, end with _FACT or _FCT suffix	End with _STAG or _STG suffix	End with _HOLD or _HLD suffix	Begin with AW_ prefix	Begin with LOOKUP_ prefix
Data endurance:	For lifetime of the Rapid Mart	For the length of execution of the job (truncated each run)	For lifetime os the Rapid Mart	Usually for the lifetime of Rapid Mart	Manually maintained
End-user reporting?	Yes	Not used	Yes, can be used	Usually not, but can be used	Typically not used
Reason for existence:	This is the final delivered data	Performance and complex operations	Usually for change data capture	Stand-alone, support, infrastructure	Performance
Example	BILLING_FACT	SO_DEL_CAND_STAG	SO_DEL_HOLD	AW_JobExecution	LOOKUP_INDEXES

Timestamps

To ensure that tables maintain proper timestamp information for data loads, complete these steps:

1. All the target tables have two columns: LOAD_DATE and LOAD_TIME.

	LOAD_DATE	LOAD_TIME
NULL values	not allowed	allowed
Data Integrator data type	datetime	varchar(8)
Oracle data type	date	varchar2(8)

2. Create two global variables at the job level:
   - $G_LOAD_DATE, with Data Integrator data type datetime
   - $G_LOAD_TIME, with Business Objects data type varchar(8)

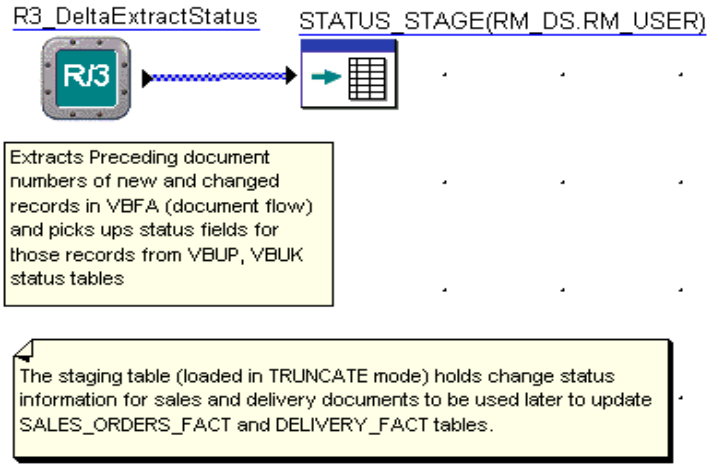
3. In the job initialization script, initialize the value of $G_LOAD_DATE to the full system date and time, up to the second. (Use the Data Integrator function sysdate().) Do not delete the time, unless your target database does not support date and time in the same data type.
4. In the job initialization script, initialize the value of $G_LOAD_TIME to the current time, up to the second. Use the Data Integrator function systime(), but convert the value to varchar to avoid warnings from the Designer.
5. In the data flow that loads a table, set the LOAD_DATE and LOAD_TIME columns as late as possible. Typically, you'll set the column to the appropriate variable in the query just before loading the table.
6. The Business Objects data type of these variables will translate to whatever is available in the target database. For example, datetime translates to date in Oracle.

Comments

Comments is a useful tool to document a Rapid Mart during its development.

Add comments in your data modeling tool that describe the tables and queries. These comments should include the mapping, followed by a colon and the description.

With Data Integrator, you can document Rapid Mart objects through their Description property or through Annotations.



Programming guidelines

This section provides programming guidelines for different types of objects:

- Datastores
- Components
- Maintenance Scripts
- Work flows
- Data flows
- Real-time data flows
- Nested queries
- Cache options
- R/3 data flows
- BO Universe

Datastores

When creating a datastore, you must set several properties. The properties vary by type of datastore.

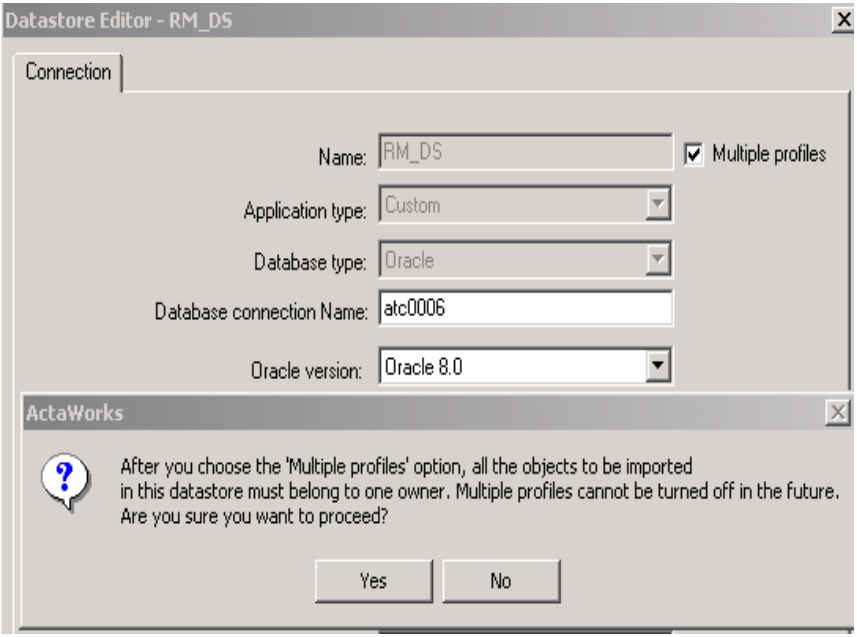
Suggested defaults for an R/3 datastore are:

Field	Suggested Value
User name	dev
Password	password
R/3 language	E-English
Execute in background (batch)	not selected
Data transport method	shared_directory

Suggested defaults for an Oracle datastore are:

Field	Suggested Value
Oracle version	Oracle 8.0
Commit size	10

You can easily switch between different database users (Oracle) or database owners (SQL Server) by creating a datastore alias (one of its properties:



Components

A Rapid Mart can contain multiple copies of the same component. To ensure that a job does not run a work flow more than once and load redundant data, select the Execute only once check box on the Properties window for each work flow that is a component.

Maintenance Scripts

The Rapid Mart Job will include all processing needed around the Data Flows like dropping/creating indexes, refreshing aggregate tables (Oracle Materialized Views, DB2 Managed Query Tables, etc), updating statistics. To hide the differences between database vendors and versions this will be done via stored procedures.

In case of a first load the pre-processing script will remove everything that hinders the DF from loading effectively but remembers those settings in LOOKUP_XXX table(s) so the post-processing script can rebuild the current status. In other words, before starting the initial load not a single manual intervention is needed

```
CREATE OR REPLACE PROCEDURE
   preprocessing_fact_table(pLoadType IN VARCHAR2,
pDropCreateIndex IN VARCHAR2, pFactTable IN VARCHAR2) IS
CURSOR cMViews(pTableName VARCHAR2) IS
SELECT mview_name
FROM USER_MVIEW_DETAIL_RELATIONS WHERE DETAILOBJ_NAME =
   pTableName AND
DETAILOBJ_TYPE = 'TABLE';
BEGIN
IF pLoadType = 'FIRST' THEN
drop_mview_logs(pFactTable);
END IF;
IF pDropCreateIndex = 'Y' OR pLoadType = 'FIRST' THEN
drop_indexes(pFactTable);
FOR rMViews IN cMViews(pFactTable) LOOP
drop_indexes(rMViews.mview_name);
END LOOP;
END IF;
END preprocessing_fact_table;


CREATE OR REPLACE PROCEDURE
   postprocessing_fact_table(pLoadType IN VARCHAR2,
pDropCreateIndex IN VARCHAR2, pFactTable IN VARCHAR2,
   pMView_Failures OUT
BINARY_INTEGER) IS
CURSOR cMViews(pTableName VARCHAR2) IS
SELECT mview_name
FROM USER_MVIEW_DETAIL_RELATIONS WHERE DETAILOBJ_NAME =
   pTableName AND
DETAILOBJ_TYPE = 'TABLE';
owner VARCHAR2(30);
BEGIN
create_indexes(pFactTable);
create_mview_logs(pFactTable);
dbms_mview.refresh_dependent(pMView_Failures, pFactTable,
   '?', NULL, TRUE,
TRUE);
FOR rMViews IN cMViews(pFactTable) LOOP
create_indexes(rMViews.mview_name);
END LOOP;
IF pLoadType = 'FIRST' OR pDropCreateIndex = 'Y' THEN
SELECT USER INTO owner FROM dual;
dbms_utility.analyze_schema(owner, 'compute');
END IF;
END postprocessing_fact_table;
```

Work flows

Data Integrator includes an automatic recovery feature that enables you to restart failed jobs. In some cases, however, you do not want to restart in the middle of work flow. For example, suppose a work flow loads two interdependent tables. You want the data in those tables to be consistent. In other words, you want both tables loaded at the same time. Therefore, when required, you must ensure that a job executes an entire work flow during automatic recovery. In these cases, select the Recover as a unit check box on the Properties window for the work flow.

Data flows

Some guidelines for creating data flows are as follows:

- No Null values in primary and foreign key columns
- There should not be any record with a null in any field marked as primary key. Further more, the key value should not be a single blank only, as when loading from SAP. All those columns are defined as “NOT NULL” in the database for optimising query response times when materialized views are used.
- No Null values in main attributes
- The most important attributes of each dimension table should not contain any null value or a single blank, like the Customer type and customer name of the customer dimension. Nulls are converted to above default parameter entries.
- Favor the exposure of source tables with joins over the use of lookup functions, whenever it makes sense.
- Verify that joins are pushed down to the database as much as possible. using Data Integrator Display Optimized SQL debug feature.

Real-time data flows

By the nature of data dealt with in real-time data flows, a large number of tables may be joined to get required information. One approach is to join all required tables in a query that retrieves the required information. This approach results in a data flow that is easy to understand and that is visually representative. However, to improve performance, you must design the query so that Data Integrator pushes data transformation to the source database.

Nested queries

When using nested structures and queries:

- Always define join conditions in the WHERE clauses.
- Remove unused structures from the FROM clause.
- When performing the unnest operation, select the **Distinct rows** check box to avoid duplicates.
- Use the Row_Generation transform to create an extra level of nesting with one-to-many relationship.
- When in a query transform you join the same table at different levels of a nested structure, create as many readers as many joins you will have.

Cache options

Avoid caching large tables where possible. See “When to cache data” on page 80 for information.

R/3 data flows

Due to restrictions in DI’s ABAP optimizer only one query should exist in each R/3 DF dealing with lots of data, like fact tables

Set the Join rank of any table participating in an R/3 data flow to 0. This will allow Data Integrator to choose the best join path when multiple tables are used.

Refer to Chapter “SAP R/3 Data Flows” of the Data Integrator Supplement for SAP R/3 for information about how to set options for R/3 data flows, such as the Cache check box. Also see “When to cache data” on page 80 for size guidelines on data caching.

BO Universe

- There is a class called “Measures” containing a subclass for all measures of a given Rapid Mart.
- In general the universe is designed in a way so that amounts of different currencies/unit of measure get not added up. The data model will contain the most common currencies/UOM already pre-calculated, but it should be possible to use all. Also it should be possible to view the data with historical correct currency conversion rates, but also with rates of today.

- Objects and subclasses always contain the main class name, so there is no “Description” it is called “Controlling Area Description”.
- Use contexts so different facts do not get combined. If different information should get combined, they are loaded in one fact table but different partitions.

RDBMS Aggregates and Indexing

Part of the data model will be useful indexing and database internal aggregation (Oracle: Materialized Views; DB2: Managed Query Tables; SQL Server: Analysis Services).

For Oracle this means that all foreign key constraints are created but disabled using the following statement:

```
ALTER TABLE CO_TRANS_FACT ADD
FOREIGN KEY (BUSINESS_AREA_ID)
REFERENCES BUSINESS_AREA (BUSINESS_AREA_ID) rely DISABLE
novalidate;
```

Bitmap Indexes are created if needed:

```
CREATE BITMAP INDEX B1_COT_COST_ELEMENT ON
CO_TRANS_FACT (COST_ELEMENT_ID, CHART_OF_ACCTS_ID) LOCAL
PARALLEL;
```

And Materialized Views:

```
CREATE MATERIALIZED VIEW M1_CO_TRANS_FACT REFRESH ON DEMAND
ENABLE QUERY REWRITE AS
SELECT
    CTRL_AREA_ID,
    COMPANY_CODE_ID,
    OBJECT_TYPE_ID,
    CO_TXN_ID,
    CO_VERSION_ID,
    VALUE_TYPE_ID,
    BUSINESS_AREA_ID,
    FUNCTION_AREA_ID,
    PERIOD,
    COST_ELEMENT_ID,
    CHART_OF_ACCTS_ID,
    FISCAL_YEAR,
    COST_CENTER_ID,
    SUM(ACTUAL_VALUE_CO_CURR) ACTUAL_VALUE_CO_CURR,
    SUM(PLAN_VALUE_CO_CURR) PLAN_VALUE_CO_CURR,
    COUNT(ACTUAL_VALUE_CO_CURR) COUNT_ACTUAL_VALUE_CO_CURR,
    COUNT(PLAN_VALUE_CO_CURR) COUNT_PLAN_VALUE_CO_CURR,
    COUNT(*)
FROM CO_TRANS_FACT
```

```
GROUP BY
    CTRL_AREA_ID,
    COMPANY_CODE_ID,
    OBJECT_TYPE_ID,
    CO_TXN_ID,
    CO_VERSION_ID,
    VALUE_TYPE_ID,
    BUSINESS_AREA_ID,
    FUNCTION_AREA_ID,
    PERIOD,
    COST_ELEMENT_ID,
    CHART_OF_ACCTS_ID,
    FISCAL_YEAR,
    COST_CENTER_ID;
```

These settings demand certain parameters set in Oracle and permissions granted when logged in as Oracle DBA:

```
ALTER SYSTEM SET OPTIMIZER_MODE=FIRST_ROWS SCOPE=SPFILE;
ALTER SYSTEM SET QUERY_REWRITE_ENABLED=TRUE;
ALTER SYSTEM SET QUERY_REWRITE_INTEGRITY=STALE_TOLERATED;

GRANT CREATE MATERIALIZED VIEW TO dwh_owner;
GRANT CREATE ANY TABLE TO dwh_owner;
GRANT COMMENT ANY TABLE TO dwh_owner;
```

Documenting a Rapid Mart

chapter 6

Documentation Template

All Rapid Mart User's Guides contain a consistent set of information. Chapters differ slightly between Analytic and Commerce Rapid Marts. Commerce Rapid Marts contain additional chapters that pertain to real-time jobs.

Topic	Chapter in Analytic Rapid Mart User's Guide	Chapter in Commerce Rapid Mart User's Guide
Introduction	Chapter 1	Chapter 1
Overview	Chapter 2	Chapter 2
Subject Areas	Chapter 3	Chapter 3
Reports	Chapter 4	Chapter 4
Real-Time Operations	not applicable	Chapter 5
Installing a Rapid Mart	Chapter 5	Chapter 6
Using the Rapid Mart	Chapter 6	Chapter 7
Technical Implementation	Chapter 7	Chapter 8
Rapid Mart Data Schema	Appendix A	Appendix A
Document Type Definitions	not applicable	Appendix B

Generally, the Introduction and Installing chapters are similar among Rapid Marts, as is most of the Using chapter. However, you must write new material for the other chapters to reflect the contents of your Rapid Mart.

Expectations for each release

Documentation expectations vary by type of release:

- Major releases — Major releases have significant changes. These releases require a new User's Guide and Release Notes, which describe new features and other changes in the release.
- Minor releases — Minor releases have additional functionality, but not significant amounts. For these releases, update the User's Guide to reflect the new functionality and create Release Notes that describe the changes. When updating the documentation, incorporate any changes previously documented only in Release Notes.

- Maintenance releases — Maintenance releases have no additional features, only bug fixes. Document the changes and fixes in Release Notes.
- Emergency bug fixes — Emergency bug fixes require a note that describes the fixed bug, referring to the bug number in your bug tracking system. The note must describe the fix in enough detail so that someone with Data Integrator experience who understands the Rapid Mart can resolve the problem with minimal or no support.

Guidelines for documentation

This section contains guidelines for the platform-independent portion of a Rapid Mart User's Guide. This section contains guidelines for these chapters:

- Overview
- Subject areas
- Reports
- Using
- Data schema

Overview

The overview should be from 3-10 pages, addressing the overall issues of how the Rapid Mart solves business problems for the customer. Required sections include:

- A one- or two-paragraph high-level description of the Rapid Mart, including discussions of business problems faced by the customer, and the functional roles of those most likely to use the Rapid Mart
- Specific business functions addressed by the Rapid Mart
- Details about the features that the Rapid Mart supports. For example, an Analytic Rapid Mart requires documentation on:
  - Types of analysis supported for each business function
  - Measures available for each type of analysis
  - Dimensions available to support these analyses
- A list of all tables in the Rapid Mart, split into groups, such as fact tables and dimension tables
- A paragraph that describes how this Rapid Mart relates to other specific Rapid Marts, both in business processes and shared tables

- Entity-relationship diagram of all tables included in the Rapid Mart, displaying table names only and using diagonal join lines

If appropriate, additional sections can be included, such as a diagram showing where the Rapid Mart fits in a business cycle.

Subject areas

For each Rapid Mart section, include the following documentation:

- One paragraph description of the Rapid Mart section, reiterating the business problems solved using the Rapid Mart.
- Entity-relationship diagram of the tables included in that view, displaying table names only and using diagonal join lines. Basically, this is the same diagram used in the Overview, but with the tables removed that do not apply to this section. Master data dimensions such as Country, Company Code, Controlling Area, etc., should be included here as appropriate. As much as possible, arrange tables in a visually elegant way. For example, for an Analytic Rapid Mart, you can arrange tables as a star schema.

  Note: If there is only one section in a Rapid Mart, do not repeat the diagram here.
- One paragraph describing the business processing that the source system does in this subject area.
- Several paragraphs describing the content of this section. Use examples to illustrate how information can be gathered from the section to solve a specific problem.
- Entity-relationship diagram of the tables included in this section, displaying table and column names, using orthogonal join lines attached to specific joined columns as much as possible. You can exclude master data dimensions, such as Country, Company Code, and Controlling Area, that are not specific to this view and that do not fit in the diagram. As much as possible, arrange tables in a visually elegant way. For example, for an Analytic Rapid Mart, you can arrange tables as a star schema.
- Discussion of specific tables. Include a general description of how the main table is created, and specific information about the grain of the data, primary keys, etc. (Also explain any dimension tables derived in non-intuitive ways.) Describe any derived columns, special transformations, filters, staging tables, or other unexpected techniques. As a general rule, if the target table is a copy of the source table or its subset, explain how it

was modeled. However, explain modeling without mentioning specific source mappings. (For example, refer to the “R/3 purchase order tables” rather than “EKKO, EKPO, and EKBE.”)

Reports

For each Rapid Mart section, include information that helps users understand the types of reports they can produce:

Sample report. Describe a business problem solved by the sample report, include a screenshot of the report, and describe how the report was created using the Rapid Mart tables.

Recommended table joins. Include a table that describes recommended joins between tables in the subject view.

Using

For this chapter, explain the usage of variables specific to this Rapid Mart and subject area.

Data schema

A listing of all tables and columns in the Rapid Mart, in the following format:

<TABLE_NAME>

Short description of table and how it is used.

Column Name	Key	Data Type	Column Description
<COLUMN_NAME>	PK FK PK/FK	VARCHAR2(XX)	Short description of column and how it is used

Multi-User Development

chapter

Overview

Rapid Marts are composed of traditional files (such as DDL scripts and sample XML messages) and Data Integrator objects (also represented by an ATL file). To maintain a history of changes and permit multiple code-lines that are simultaneously manipulated by several developers, you can use both traditional code management software (for example, VSS) and the Data Integrator multi-user development environment.

The multi-user environment within Data Integrator is similar to Microsoft VSS. In Data Integrator, you maintain objects in a central repository and check out copies of the objects to a local repository, where you work on them. Data Integrator lets you compare two objects, enabling you to identify differences between objects in different repositories. For more information about the multi-user environment, see Chapter "*Multi-User Development*" of the *Data Integrator Designer Guide.*

Development of a new Rapid Mart

You develop a new Rapid Mart in a local repository. Existing Rapid Marts must reside in a central repository linked to your local repository. Often, you can reuse some components from existing Rapid Marts. You must retrieve the latest versions of any needed components from the central repository into the local repository.

Follow these steps when developing a new Rapid Mart:

- Create or adjust your datastores to point to your new Rapid Mart project (for example, PP, SM, etc.).
- Create one or more projects that will form the Rapid Mart.
- Determine what components you will reuse from existing Rapid Marts and get the latest version of those objects.
- Create new objects.
- When done creating and testing objects, add those objects to the central repository.
- Verify that no changes were made to the reused components, which you retrieved from the central repository using the get latest command. If any components were modified, import the latest versions again and check that these modified components function in your Rapid Mart.

Modification of an existing Rapid Mart

When modifying components in an existing Rapid Mart, you must check the components out of the central repository, edit them in the local repository, and then check the changed component back in to the central repository. Before modifying an existing component, however, you should notify other Rapid Mart developers.

Synchronization

The central repository stores all the objects in your Rapid Marts. Whenever you or someone on your team develops a new Rapid Mart, you get the latest version of needed components from the central repository and copy them into your local repository. You use these copied versions of the components in your new Rapid Mart. Meanwhile, if someone changes any of these components, you must re-import them from the central repository and make sure that the new versions work with your Rapid Mart before checking the new Rapid Mart into the central repository.

Data Modelling Issues

chapter 8

Normalised versus Dimensional schemas

Compared to dimensional schemas, normalized schemas can capture semantic changes more easily and usually mimic transactional systems better. Often, a normalized data model is more appropriate for data integration activities, because the normalized model captures more of the data's intricate properties. This makes a normalized model more conducive to "correctness" (less data redundancy), which is important for both translation and integration, especially when there are semantic changes.

Consider two situations: customer hierarchy information and sales order updates. Usually, customer information embeds several flattened hierarchies. Translating the fields involved in a hierarchy can be quite cumbersome in a non-normalized schema. Updating hierarchies and maintaining multiple versions of hierarchies can also present more challenges in a non-normalized schema.

Sales orders, when not modeled in a transactional fashion, might contain de-normalized pieces of data, such as "total order value," at different levels, such as line items. Updating the quantity of a certain line item, therefore, triggers the update of all line items belonging to that order, since the total order value needs to be changed.

A dimensional schema, on the other hand, is very usable. To achieve a balance between the two types of schemas, you can create an intermediate, normalized schema, then build a dimensional model from that normalized schema. The final model could address usability and other considerations, like performance and history preservation.

International considerations

Your data model needs to consider several issues to support internationalization:

- Multi-language support
- Multi-language support in real time jobs
- Multi-currency support
- Multiple time-zone support
- Multi-byte codepages

Multi-language support

When creating a Rapid Mart for customers from multiple countries, you will want to store descriptive messages in each user's native language. A Rapid Mart can extract multi-language descriptions from the source system (SAP R/3, for example). You can let the end-user applications select the preferred language for displaying the information.

When a Rapid Mart object contains descriptive fields (for example, material, customer, sales order header), store those descriptions in the base table in the default language. This improves performance in single-language situations and allows for default information when values are missing for a particular language.

As long as the number of additional languages is small add those as fields of the base table as well, suffix the column with a number like DESC (primary language), DESC_1, DESC_2 and create new global variables $G_LANGUAGE_1, etc.

With this design, multi-language support is optional and extensible for the Rapid Mart.

Multi-language support in real time jobs

To support descriptions in multiple languages in real-time jobs, you must consider two types of messages:

- Messages returned by your source system real-time APIs

  Examples of this type of message are the SAP R/3 messages returned from a BAPI call. Data Integrator 4.3 and higher can pass the language parameter into a BAPI call. The BAPI will return its messages in the appropriate language. If the language is not defined in an incoming message, the BAPI will return its message in the default language set during the creation of its associated datastore. You can set a different default by using a global variable, such as $G_LANGUAGE, which is set in the initialization script, whenever designing a real-time job.

- Rapid Mart built-in messages

  When processing data in real-time data flows, a script function can generate appropriate messages through a standard interface, such as the Data Integrator AW_MapMessage function included in the Customer Order Rapid Mart. By default, the function returns all messages to the requesting application in a single default language.

AW_MapMessage function

To support other languages, you must:

- Translate all messages in this function to those languages.
- Edit the function and add new statements that translate each message:

```
…
else if($LANGUAGE_ID = 'D')
BEGIN
…
END
```

- Either enhance your input message formats to receive the language parameter as part of each message, or start specific RTDFs to handle specific languages and have the front-end application make the language decision.

When extending this function, you can follow your source system conventions for internal language abbreviations (for example, English is E in SAP R/3), or you can use the international standard. Using the source system convention makes it easier to use an external function call (such as BAPI call for SAP R/3), since you can pass the language identification directly.

Multi-currency support

Because enterprises operate in a global economy, you must often deal with multiple currencies when loading your Rapid Mart. For example, you might have purchase orders issued in US dollars and EUR. Traditionally, ERP systems store monetary values in at least two different currencies: the local currency in which the entry was made (for example, EUR), and a company - wide currency used for financial purposes (for example, U.S. dollars).

However, an application using your Rapid Mart might request monetary values in yet another currency (for example, the Euro). To respond to such requests, you must capture enough information from your source systems to convert currency. You must also decide whether to use your source system's currency conversion algorithms or whether to define your own algorithms, based on business requirements.

Multiple time-zone support

If your Rapid Mart will support multiple users in different time-zones, you must consider multiple time-zone support. For example, suppose your Rapid Mart and the ERP system are in California and a user in Paris, France enters an order. The transaction time is based on the time where the ERP system is located. If the user were working early in the morning, the time in California —

and therefore, the recorded transaction time—might be late the previous night. If the user attempts to verify all transactions placed on the current day shortly after placing his order, the system might not present the just-placed order unless you convert the requested date according to a time-zone filter.

You can support multiple time zones with Rapid Mart data and with Rapid Mart logic. For example, you provide support with data if you store information pertaining to the time-zone where a transaction originated in addition to the ERP-based time-zone. Similarly, you provide support with logic if you convert dates with a time zone filter before retrieving data.

A traditional solution to this problem is to store and process all date-related information in a single time zone (for example, GMT), and converting before presenting to the user if required.

Multi-byte codepages

To support multiple languages, you might need to support multi-byte languages. A multi-byte language is one that requires more than one byte for each character. For example, to represent 20 characters, you might require 45 bytes.

Facilitating multi-byte codepages might simply require that you allow for more space when designing tables and data flows. However, an application can become more complex if it requires string matching or string manipulation for different codepages.

Using surrogate keys

Surrogate keys are internal keys, usually numeric, that substitute for an object's natural keys in a certain source system. Surrogate keys uniquely identify an instance of an object.

General data warehousing practices recommend using surrogate keys to uniquely identify all dimensions of a star-based schema. Using surrogate keys is recommended because natural keys, also called "production keys," are not always unique across time. For example, a company might discontinue a product one year, and then launch a different product with the same code the following year. In this case, the history of sales that the data warehouse maintains could render wrong results when accumulating values per product, over time.

Surrogate keys are particularly useful for handling slowly changing dimensions. Slowly changing dimensions are objects with just a few attributes that change over time and that you need to track independently. For example,

if a person changes his or her address, you must maintain both the new and old information, so that you can query the volume of sales per region over time and get consistent results.

However, in some situations you can use natural keys to identify dimensions. For example, if your source system provides natural keys that offer significant control, and those using your Rapid Mart do not require that you capture slow changes in dimensions, using natural keys is unlikely to result in any problems.

Further more, Rapid Marts are build upon one and only one ERP systems thus the keys can be trusted in most cases. As a result Rapid Marts use natural keys in general, surrogate keys in case the source data cannot be trusted or the primary key got lost.

Staging data

Staging information is usually done for one of the following reasons:

- To improve performance due to technical limitations (network bandwidth, optimization limitations, etc.)
- To execute complex operations in more manageable steps
- To preserve control information from previous loads in order to detect changes to the data (for example, capturing physical deletions in the source system)

However, using stage tables reduces performance as the data has to additionally be written into the stage table and read from it. And there is the question of maintainability too.

When staging data, consider how long you will keep the data that you stage. Most often, keeping the data for the life of a job or even the work flow is sufficient. Other times, you might need to keep the staged data until after the next run. If you are unable to reconstruct your target database by re-reading source information, you might consider backing up staged data.

Typically, staged data is not used for external purposes, though the data might be exposed during administrative tasks. For example, suppose your Rapid Mart stores keys from the previous load in a staging table in order to identify deletions. In this case, your Rapid Mart administrator might want to query the staging table that contains those keys before actually running your jobs.

For information about the characteristics of staging tables, see “Table types” on page 47

Data Movement Algorithms

chapter 9

Changed-data capture

When designing a Rapid Mart to implement changed-data capture (CDC), you must consider:

- Header and detail synchronization
- Capturing physical deletions

Header and detail synchronization

Typically, source systems keep track of header and detail information changes in an independent way. For example, if a line item status changes, its “last modified date” column is updated, but the same column at the order header level is not updated. Conversely, a change to the default ship-to address in the order header might impact none of the existing line items.

In some instances, however, your source system might not consistently update those tracking columns, or you might not have access to such information (for example, when rows are physically deleted). In these cases, you may choose to extract all header and detail information whenever any changes occur at the header level or in any individual line item.

To extract all header and detail rows when any of these elements have changed, use logic similar to this SQL statement:

```
SELECT …
FROM HEADER, DETAIL
WHERE
    HEADER.ID = DETAIL.ID AND
    (HEADER.LAST_MODIFIED
        BETWEEN $G_SDATE AND $G_EDATE
        OR
        DETAIL.LAST_MODIFIED
        BETWEEN $G_SDATE AND $G_EDATE)
```

For some RDBMSs, this WHERE clause is not well optimized, and might cause serious performance degradation. You might opt to relax that clause by removing one of the upper bounds, such as in:

```
…
WHERE HEADER.ID = DETAIL.ID AND
    (HEADER.LAST_MODIFIED
        BETWEEN $G_SDATE AND $G_EDATE
        OR
        DETAIL.LAST_MODIFIED >= $G_SDATE)
---
```

This might retrieve a few more rows than originally intended, but it might improve the final performance of your system, while not altering the result of your target database.

Capturing physical deletions

When your source system allows rows to be physically deleted, your Rapid Mart should include logic to update your target database correspondingly. There are essentially five ways to do this:

1. Scanning a log of operations — If your system logs transactions in a readable format or if you can alter the system to generate such a log, then you can scan that log to identify the rows you need to delete.
2. Performing a full refresh — Simply reload all of the data, therefore fully synchronizing the source system and the target database.
3. Performing a partial refresh based on a data-driven time-window — For example, suppose that the source system only allows physical deletion of orders that have not been closed. If the first non-closed order in your source table occurs 6 months ago, then by refreshing the last 6 months of data you are guaranteed to have achieved synchronization.
4. Performing a partial refresh based on a business-driven time-window — For example, suppose that the business that the Rapid Mart supports usually deletes orders shortly after creating them. In this case, refreshing the last month of orders is appropriate to maintain integrity.
5. Checking every order that could possibly be deleted — You must verify whether any non-closed order has been deleted. To be efficient, this technique requires you to keep a record of the primary keys for every object that is a candidate for deletion.

When physical deletions of detail information in a header-detail relationship are possible (for example, removing line items from an existing order), then you must capture these physical deletions when synchronizing header and detail information.

Reverse pivoting

The Data Integrator Pivot transform creates a new row for each value in a column that you identify as a pivot column—that is, the transform moves horizontal data vertically. To reverse this transform, you can use SQL.

Suppose that you have the following source table:

MATERIAL	JAN	FEB	MAR
tools	123	456	789
toys	2	6	
games	-1	-2	-3

Using the Pivot transform on this table, produces the following results:

MATERIAL	SEQ	MONTH	VALUE
tools	1	JAN	123
tools	2	FEB	456
tools	3	MAR	789
toys	1	JAN	2
toys	2	FEB	
toys	3	MAR	6
games	1	JAN	-1
games	2	FEB	-2
games	3	MAR	-3

Using the Reverse Pivot transform on this table, produces the following resultes:

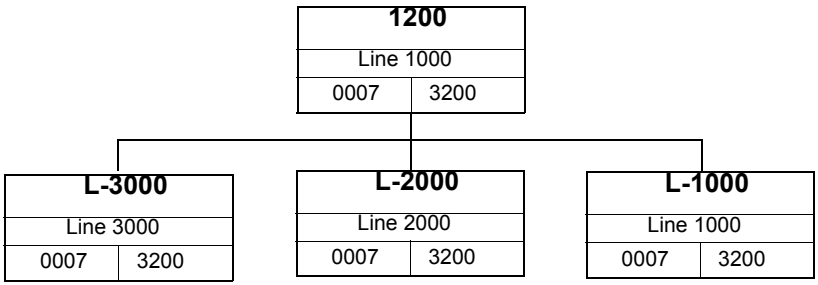
MATERIAL	JAN	FEB	MAR
tools	123	456	789
toys	2	0	6
games	-1	-2	-3

Most likely, Data Integrator will request that the underlying RDBMS complete this operation. This ensures adequate performance, particularly if the appropriate indexes are defined in the tables.

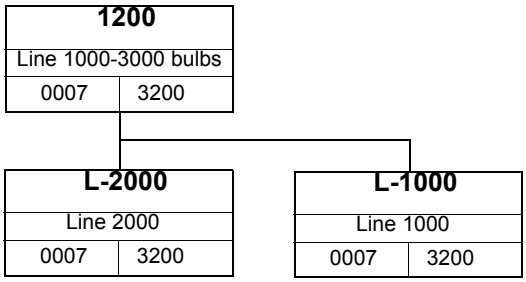
Multiple hierarchies

The Hierarchy_Flattening transform assumes parents and children belong to a single hierarchy when flattening the relationship. Often, however, multiple hierarchies exist, differentiated by a primary key. You cannot pass a primary key to the Hierarchy_Flattening transform. Instead, you must create one by concatenating the primary key field to the parent and children columns. After flattening the hierarchy, you must extract the primary key field from the parent and child leaf columns that the transform generates, and return the keys to their original values.

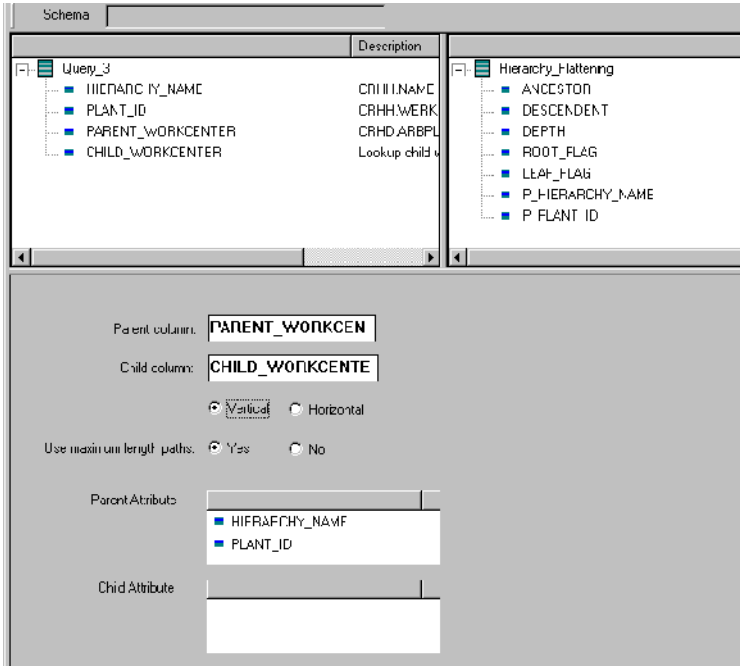
For example, suppose you have a work center relationship defined by hierarchy 1200 in plant 3200. This hierarchy has three work centers—L-3000, L-2000, and L-1000—that roll up to a single work center, Line 1000, within plan 3200.



Suppose plant 1200 also has hierarchy 1200. At this plant, the hierarchy has two work centers—L-2000 and L-1000—that roll up to work center Line 1000 -3000 bulbs. Although the hierarchy is the same—hierarchy 1200—the parent work center changes because this hierarchy belongs to plant 1200.



In Data Integrator, you can set the Hierarchy_Flattening parameters to vertically flatten this hierarchy without creating concatenated keys.



However, the transform only selects the first valid record it finds. Therefore, the flattened hierarchy that results only contains data for plant 3200; it does not contain data for plant 1200. Though the input data contains five descendents for hierarchy 1200 in both plants, the transform only selects the first one, which contains the descendents from plant 3200.

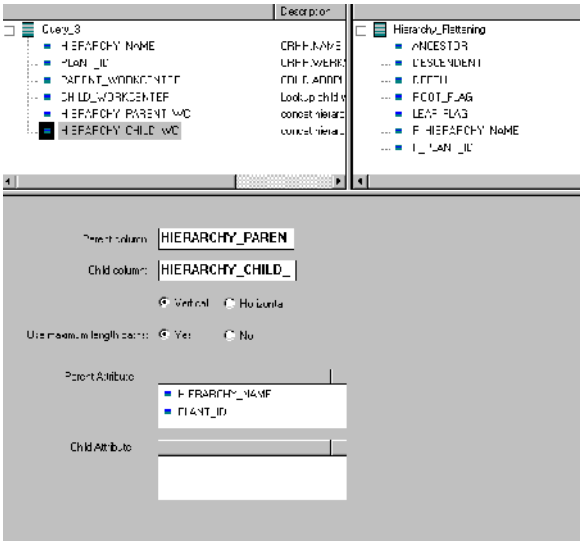
P Plant Id	Ancestor	Descendent	Depth	Root Flag	Leaf Flag
3200	1200	L-1000	1	0	
3200	1200	L-2000	1	0	
3200	1200	L-3000	1	0	

To avoid this result, create new parent and child columns by concatenating the plant ID to the parent and child work centers. For example:

```
HIERARCHY_PARENT_WC = lpad(PLANT_ID, 4, 0) ||
lpad(PARENT_WORKCENTER, 8,0)

HIERARCHY_CHILD_WC = lpad(PLANT_ID, 4, 0) ||
lpad(CHILD_WORKCENTER, 8, 0)
```

In Data Integrator, set the Hierarchy_Flattening parameters to vertically flatten the hierarchy using the concatenated keys.



The flattened hierarchy contains the following data:

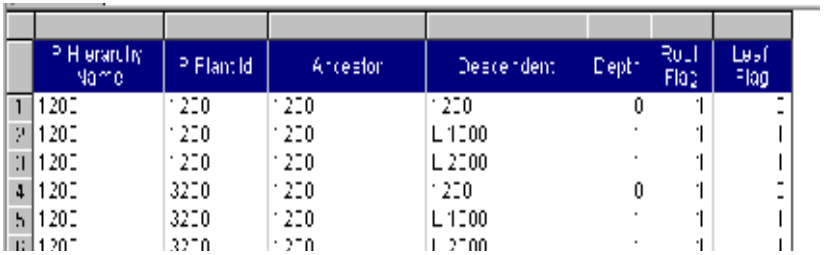
	P Hierarchy Name	P Plant Id	Ancestor	Descendent	Depth	Root Flag	Leaf Flag
1	1200	1200	1200 0000 0000 0	1200 0000 0000 0	0	1	0
2	1200	1200	1200 0000 0000 0	1200 0000 1000	1	1	1
3	1200	1200	1200_0000_2000	1200_0000_2000	0	1	1
4	1200	3200	3200000020	3200000020	0	1	0
5	1200	3200	3200000020	3200001000	1	1	1
6	1200	3200	3200000020	3200002000	1	1	1
7	1200	3200	3200000020	3200003000	1	1	1

These results are correct, but you must reverse the concatenation and remove the plant from the ancestor and descendent fields. You can create mappings to remove the plant ID:

```
ANCESTOR =ltrim(substr(Hierarchy_Flattening.ANCESTOR, 5,8),
    '0')
DESCENDANT =ltrim(substr(Hierarchy_Flattening.DESCENDENT,
    5,8), '0')
```

Note: If you do not have numeric keys, you can pad with leading blanks instead of 0s and use the ltrim function to remove the blanks.

The results are now correctly flattened within each plant.



	P Hierarchy Name	P Flan:Id	Ancestor	Descendent	Depth	Root Flag	Leaf Flag
1	120[illegible]	[illegible]220	[illegible]220	[illegible]220	0	1	[illegible]
2	120[illegible]	[illegible]220	[illegible]220	L1200	[illegible]	1	1
3	120[illegible]	[illegible]220	[illegible]220	L2200	[illegible]	1	1
4	120[illegible]	3220	[illegible]220	[illegible]220	0	1	[illegible]
5	120[illegible]	3220	[illegible]220	L1200	[illegible]	1	1
6	120[illegible]	3220	[illegible]220	L2200	[illegible]	1	1

You can use the same approach for horizontal flattening. Here all levels of descriptions and IDs are filled, if nothing can be found the element of the level above is taken. In other words the mapping “LEVEL_n = nvl(LEVEL_n, nvl(LEVEL_n-1, nvl(LEVEL_n-2, ….)))” is used. Also the primary key of this table has to be the leaf element thus the flow has to make sure only one root element per leaf level is loaded. If that cannot be automated e.g. by getting the Standard Hierarchy information, the primary key has to exist, is disabled and in the installation guide it has to be told that this is left to the person configuring the Rapid Mart. When building Hierarchies consider this and other solutions to report on those hierarchies using BO

Performance considerations

This section discusses several performance considerations you must make when designing a Rapid Mart:

- When to cache data
- Determining join ranks
- Function-based lookups versus explicit joins

When to cache data

BusinessObjects recommends that you apply table caching only for tables less than 2 MB. You can calculate the approximate size of a table with the following formula:

Determining join ranks

For R/3 data flows, the join order (rank order) of tables participating in a join depends on the version of SAP R/3. From release 4.0 onwards, OpenSQL introduces the opportunity for Data Integrator to push down more complex operations (most notably joins) to the underlying database. Therefore, for

releases prior to SAP R/3 4.0 (for example, 3.1h), specifying the correct rank order for table joins is important. Regardless of the type of data flow—batch or R/3—the larger the table size, the larger the rank order should be. Therefore, if table A has 1 million rows and table B has 1,000 rows, table A should be ranked higher than table B. Also note that:

- If table A has an index on the join predicate, then inverting the order is usually not very significant.
- If the inner table (B in the example above) is reasonably small (say, less than 5 blocks), then having an index on the join predicate may be even slower than full-table scans.

Note: If you specify the join rank order (that is, the parameters are non-zero), then Data Integrator will generate ABAP to execute the requested join order, which might not be what you want for higher versions of SAP R/3.

Function-based lookups versus explicit joins

Although you can use lookup functions inside Data Integrator queries, BusinessObjects recommends that you expose the table being looked up as a source table in the data flow diagram, and use an outer join (if necessary) in the query to look up the required data. This technique has several advantages:

- You can determine what the data flow does directly from the diagram, making the data flow easier to maintain
- Data Integrator can push the execution of the join down to the underlying RDBMS (even if you need an outer join)
- You can retrieve multiple columns at the same time
- You can take advantage of in-memory caching

This technique also has some disadvantages:

- Workspace can become cluttered if there are too many objects in the data flow . Therefore use lookups for all the small key-to-text conversions. With lookup_ext you can control the record to return in case multiple are found, e.g. the latest version
- There is no option to use DEMAND_LOAD caching, which is useful when looking up only a few values in a very large table.

Extracting Data from SAP R/3

appendix

A

Overview of SAP R/3

This section discusses key features about SAP R/3:

- SAP R/3 type of data structures
- Useful SAP R/3 transactions

SAP R/3 type of data structures

Depending on the expected size of an entity's data, SAP R/3 uses different types of data structures:

- Transparent table — A standard database table recognized by any RDBMS; used for master or transactional data.
- INTTAB 3 — An internal SAPR/3 structure; used for temporary storage of data from database tables for ABAP programs at a runtime.
- Cluster table — A proprietary SAP R/3 object; a flat table with "hierarchical" appearance. It consists of multiple regular database tables with a similar key (obsolete in SAP R/3 version 3.0).
- Pool table — A proprietary SAP R/3 object; a bulk storage of many logical tables in a regular database table. Essentially, this is a vertical union of multiple tables (obsolete in SAP R/3 version 3.1).
- View — A standard database view; a hierarchy of transparent tables linking transaction or master data.
- Append structure — A proprietary SAP R/3 object used when expansion of existing transparent tables is needed. Instead of adding similar fields to multiple transparent tables, an SAP R/3 user can create an append structure and point to it from those tables.

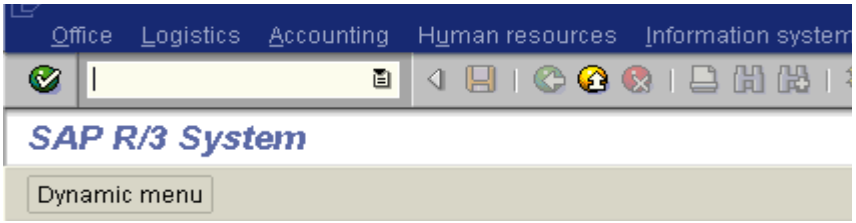
Data Integrator stores information about R/3 table types in a table attribute called SAP_Table_Class. Data Integrator uses this information when generating ABAP code. (You—the developer—do not need to know these details; Data Integrator handles this automatically.) For more information on Data Integrator R/3 table attributes, see Chapter "*SAP-Oriented Reference Information*," of the *Data Integrator Supplement for SAP R/3*.

Useful SAP R/3 transactions

Navigation in SAP R/3 is much faster if you use transaction codes.

To get to a desired screen, go to a transaction editor and type:

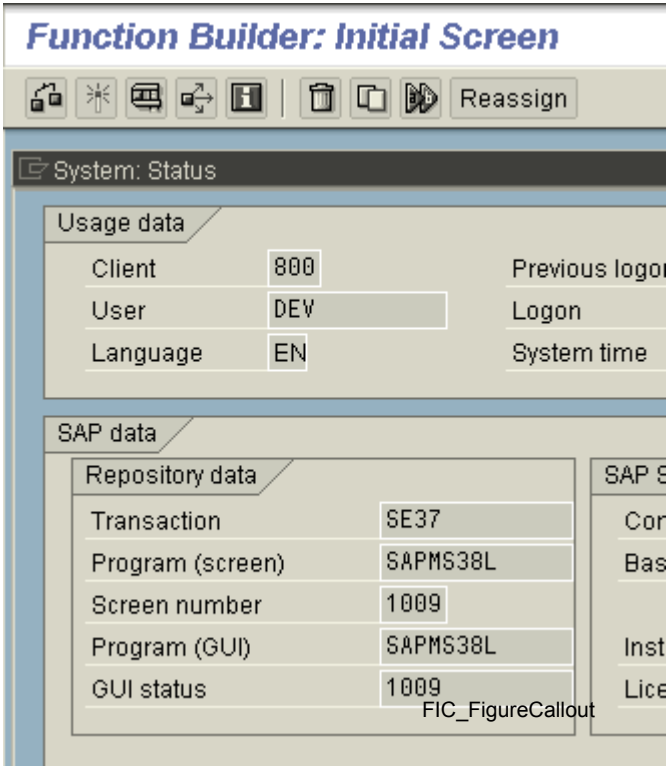
```
/n<transaction code>
```



Useful SAP R/3 transactions include:

- BAPI — BAPI interface display (in SAP R/3 version 4.0 and higher)
- SB10 — BAPI interface display (in R/3 version 3.1)
- SE11 — Data dictionary maintenance
- SE16 — Table row count (number of entries)
- SE17 — Data dictionary display
- SE37 — Function module maintenance
- SE38 — ABAP program maintenance
- SM12 — Unblock table
- SM36 — Batch job maintenance
- SM37 — Batch job overview
- SM50 — Process overview
- SU02 — Authorization profile
- WE05 — IDOC list display
- WE60 — IDOC type display
- WE64 — IDOC metadata display

You can create your own list of frequently-used transactions. When you have a desired screen displayed, go to the SAP R/3 menu bar and select **System > Status**. The transaction number for this screen will be displayed in the **Transaction** field on the **Repository data** tab, under **SAP data**.



How Data Integrator interacts with SAP R/3

The Data Integrator Supplement for SAP R/3 contains detailed information on how Data Integrator interacts with SAP R/3. This section lists particular chapters that you can refer to for more information.

Browsing and importing SAP R/3 metadata

For information about browsing metadata information and importing metadata, see Chapter "*SAP R/3 Datastores*."

Working with SAP R/3 hierarchies

SAP R/3 stores object hierarchy dependencies using different techniques:

- Simple one-to-many hierarchy — A parent is referenced in a record where its child is a primary key (for example, equipment hierarchy, functional location hierarchy). For more information about this kind of hierarchy, see "*Hierarchy_Flattening*" on the *Data Integrator Reference Guide*.
- Complex many-to-many or one-to-many relationships — SAP R/3 supports hierarchy trees through a notion of "sets." They are used for hierarchies in SAP R/3 Cost and Accounting modules and supported by Data Integrator. For more information, see Chapter "*SAP R/3 Datastores*" and Chapter "*SAP R/3 Data Flows*" of the *Data Integrator Supplement for SAP R/3*.

ABAP generation and RFC connection

Refer to:

- Chapter "*SAP R/3 Data Flows*"
- Chapter "*Real-Time and SAP R/3*"

R/3 data flow

For information about connecting to SAP R/3, security, and ABAP performance optimization, see Chapter "SAP R/3 Data Flows".

Job execution — modes, scheduling, monitoring in R/3

For information about executing jobs—the various modes, scheduling, and monitoring in SAP R/3—see:

- Chapter "SAP R/3 Datastores"
- Chapter "Executing Batch Jobs in the SAP R/3 Environment"

Note: To monitor the status of Data Integrator batch job execution using the SAP R/3 client, use an SAP R/3 screen associated with transaction SM37 (batch job overview).

BAPIs used in Rapid Marts

Existing Business Objects Rapid Marts use several BAPIs to interact with SAP R/3, including:

- BAPI_SALESORDER_CREATEFROMDATA
- BAPI_SALESORDER_CREATEFROMDAT1

- BAPI_SALESORDER_CHANGE
- BAPI_SALESORDER_SIMULATE
- BAPI_SALESORDER_GETSTATUS
- BAPI_MATERIAL_AVAILABILITY

SAP R/3 has good documentation for BAPIs in version 4.6.B and higher. The easiest way to access this documentation is from the Function Builder screen (transaction SE37).

The availability of BAPIs varies significantly between different versions of SAP R/3. Some BAPIs created in earlier versions became obsolete in subsequent versions (for example, BAPI_SALESORDER_CREATEFROMDATA is not maintained starting from version 4.0.B and is removed from the source code in version 4.6.C).

For detailed information about how to call BAPIs from Data Integrator for real-time operations, see “SAP R/3 RFC and BAPI function calls” of the Data Integrator Supplement for SAP R/3.

IDocs

For basic information about SAP R/3 IDocs, see “Sources in real-time data flows” on the Data Integrator Supplement for SAP R/3.

IDocs are useful when the data platform must immediately capture certain events that happen in the back-office application. For example, suppose you have a web-enabled HR self-service application based on a cached data platform. When an employee is terminated, the business might mandate that the employee’s access to the application be terminated immediately. You can handle this situation by having SAP R/3 administrators turn on the IDoc related to that particular transaction (changing the status of personnel), and running a specific real-time data flow that updates the data cache in real-time when the message is received.

This approach requires turning on IDocs in a production SAP R/3 system. This requires the R/3 engine to complete extra work during transactions. Therefore, use this option cautiously, only after carefully evaluating consequences with your SAP R/3 team.

SAP R/3 Certification Guide

Possible changes in SAP R/3 table schemas

With every new release, SAP tends to extend the length of some of table fields of char data type. To avoid data overflow, the width of every Rapid Mart target table field should be compatible with the width of the SAP R/3 tables in the latest release.

▶ **To perform a table schema comparison:**

1. Run Calculate Dependency metadata report to know which SAP R/3 table fields are used by a Rapid Mart.
2. Determine which target table fields are affected by the SAP R/3 table fields.
3. Verify if the widths of those SAP R/3 table fields has changed. (There are multiple ways to identify fields that have changed. For example, you can do a text diff on schema dumps.)
4. Update your target table schema and extraction logic to accommodate necessary changes of a field width.

With new releases of R/3, SAP often changes field characteristics, pertaining to NULL values. This characteristic can change from NULL to NOT NULL or from NOT NULL to NULL. Changing this characteristic should not affect Rapid Marts. Business Objects recommends that you run Rapid Marts with the Data Integrator server option **Convert R3 null to null** turned off (this is the default setting). This setting allows Data Integrator to insert a space ' ' for unknown SAP R/3 data of the char data type and a zero 0 for numeric data types.

SAP R/3 table descriptions change frequently also. Business Objects generally ignores these changes because they do not affect the Rapid Mart functionality.

In some cases, SAP adds additional fields to the primary key. To avoid primary key violation errors, you must modify the extraction logic and target table schemas to accommodate additional elements of primary keys.

Possible substitution of existing tables

Sometimes SAP marks certain tables as “not in use anymore” and starts using new tables for the same purposes. In this case, you must change extraction and transformation logic to pull data from a new table.

Possible changes in SAP R/3 table class

Occasionally, SAP changes a table class between releases of R/3. For example, CDHDR table is a cluster table in SAP 3XX releases, but becomes a transparent table in 4XX releases. Data Integrator generates different ABAP for different table classes. You must find such changes and re-import changed tables into the Data Integrator repository

Note: Instead of changing IM_REPO user name in the TableComparison.atl file you can create an alias for the datastore and point to the repository to be compared.

Business Objects Information Resources

Documentation and information services

Business Objects offers a full documentation set covering its products and their deployment. Additional support and services are also available to help maximize the return on your business intelligence investment. The following sections detail where to get Business Objects documentation and how to use the resources at Business Objects to meet your needs for technical support, education, and consulting.

Documentation

You can find answers to your questions on how to install, configure, deploy, and use Business Objects products from the documentation.

What's in the documentation set?

View or download the *Business Objects Documentation Roadmap*, available with the product documentation at http://www.businessobjects.com/support/.

The Documentation Roadmap references all Business Objects guides and lets you see at a glance what information is available, from where, and in what format.

Where is the documentation?

You can access electronic documentation at any time from the product interface, the web, or from your product CD.

Documentation from the products

Online help and guides in Adobe PDF format are available from the product Help menus. Where only online help is provided, the online help file contains the entire contents of the PDF version of the guide.

Documentation on the web

The full electronic documentation set is available to customers on the web from support web site at: http://www.businessobjects.com/support/.

Documentation on the product CD

Look in the docs directory of your product CD for versions of guides in Adobe PDF format.

Send us your feedback

Do you have a suggestion on how we can improve our documentation? Is there something you particularly like or have found useful? Drop us a line, and we will do our best to ensure that your suggestion is included in the next release of our documentation: documentation@businessobjects.com.

Note: If your issue concerns a Business Objects product and not the documentation, please contact our Customer Support experts. For information about Customer Support visit: http://www.businessobjects.com/support/.

Customer support, consulting and training

A global network of Business Objects technology experts provides customer support, education, and consulting to ensure maximum business intelligence benefit to your business.

How can we support you?

Business Objects offers customer support plans to best suit the size and requirements of your deployment. We operate customer support centers in the following countries:

- USA
- Australia
- Canada
- United Kingdom
- Japan

Online Customer Support

The Business Objects Customer Support web site contains information about Customer Support programs and services. It also has links to a wide range of technical information including knowledgebase articles, downloads, and support forums.

http://www.businessobjects.com/support/

Looking for the best deployment solution for your company?

Business Objects consultants can accompany you from the initial analysis stage to the delivery of your deployment project. Expertise is available in relational and multidimensional databases, in connectivities, database design tools, customized embedding technology, and more.

For more information, contact your local sales office, or contact us at:

http://www.businessobjects.com/services/consulting/

Looking for training options?

From traditional classroom learning to targeted e-learning seminars, we can offer a training package to suit your learning needs and preferred learning style. Find more information on the Business Objects Education web site:

http://www.businessobjects.com/services/training

Useful addresses at a glance

Address	Content
Business Objects product information http://www.businessobjects.com	Information about the full range of Business Objects products.
Product documentation http://www.businessobjects.com/support	Business Objects product documentation, including the Business Objects Documentation Roadmap.
Business Objects Documentation mailbox documentation@businessobjects.com	Send us feedback or questions about documentation.
Online Customer Support http://www.businessobjects.com/support/	Information on Customer Support programs, as well as links to technical articles, downloads, and online forums.
Business Objects Consulting Services http://www.businessobjects.com/services/consulting/	Information on how Business Objects can help maximize your business intelligence investment.
Business Objects Education Services http://www.businessobjects.com/services/training	Information on Business Objects training options and modules.