Business Objects owns the following U.S. patents, which may cover products that are offered and sold by Business Objects: 5,555,403, 6,247,008 B1, 6,578,027 B2, 6,490,593 and 6,289,352.

Business Objects owns the following U.S. patents, which may cover products that are offered and licensed by Business Objects: 5,555,403; 6,247,008 B1; 6,578,027 B2; 6,490,593; and 6,289,352. Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Xcelsius, Crystal Decisions, Intelligent Question, Desktop Intelligence, Crystal Enterprise, Crystal Analysis, WebIntelligence, RapidMarts, and BusinessQuery are trademarks or registered trademarks of Business Objects in the United States and/or other countries. All other names mentioned herein may be trademarks of their respective owners.

© 2007 Business Objects. All rights reserved.

Business Objects products in this release may contain redistributions of software licensed from third-party contributors. Some of these individual components may also be available under alternative licenses. A partial listing of third-party contributors that have requested or permitted acknowledgments, as well as required notices, can be found at: http://www.businessobjects.com/thirdparty

April 26, 2007
## Contents

### Chapter 1 Overview
- Migration overview .............................................. 8
- About this document ........................................ 8
- Audience and assumptions ................................... 9
- More Data Integrator product documentation .......... 9

### Chapter 2 Migration basics
- About this chapter ............................................. 12
- Development phases .......................................... 12
  - Design phase ................................................. 13
  - Test phase .................................................. 13
  - Production phase ......................................... 13
- Migration mechanisms and tools ............................ 14
  - Which mechanism is best? ................................. 15
  - Export/import migration .................................. 15
  - Multi-user migration ....................................... 16

### Chapter 3 Preparing for migration
- About this chapter ............................................. 18
- Naming conventions for migration ........................ 18
  - Connections to external datastores ..................... 19
  - Directory locations ....................................... 20
  - Schema structures and owners .......................... 20
- Datastore and system configurations ...................... 21
  - Datastore configurations and migration ............... 22
  - Multiple configurations in multi-user environments .. 24
- Command line login to the Designer ...................... 25
Chapter 4 Export/import 27
Overview of export/import ........................................... 28
Exporting/importing objects in Data Integrator .................. 28
The Export editor ...................................................... 28
Exporting objects to another repository ......................... 31
Exporting objects to a file ......................................... 32
Exporting a repository to a file .................................. 33
Importing from a file .................................................. 33
Command line options to export objects to an XML file ....... 34
Removing obsolete repository contents ......................... 35
Backing up repositories ............................................. 36
Maintaining Job Server performance ............................ 36

Chapter 5 Multi-user development 39
About multiple users ............................................... 40
Central versus local repository .................................. 40
Data Integrator and multiple users ............................... 41
Security and the central repository .............................. 43

Chapter 6 Multi-user environment setup 45
Overview of multi user setup ...................................... 46
Create a nonsecure central repository ......................... 46
Define a connection to a nonsecure central repository ....... 47
Activating a central repository .................................. 47

Chapter 7 Implementing Central Repository Security 51
About this chapter ................................................... 52
Overview .............................................................. 52
Group-based permissions .......................................... 52
Permission levels ..................................................... 53
Process summary .................................................... 53
Creating a secure central repository ......................... 54
Adding a multi-user administrator (optional) .................. 55
Setting up groups and users ..................................... 55
Defining a connection to a secure central repository ....... 55
Chapter 8 Working in a multi-user environment

Overview of multi-user tasks .................................................. 60
Filtering ................................................................. 60
Adding objects to the central repository .................................. 61
Checking out objects ......................................................... 63
  Check out single objects or objects with dependents ................. 63
  Check out single objects or objects with dependents without replacement 65
  Check out objects with filtering ........................................... 66
Undoing check out ............................................................... 66
Checking in objects ............................................................ 67
  Checking in single objects, objects with dependents .................. 68
  Checking in an object with filtering ....................................... 69
Labeling objects ............................................................... 69
Getting objects ............................................................... 71
Comparing objects ............................................................. 72
Viewing object history ......................................................... 72
Deleting objects ............................................................... 74

Chapter 9 Migrating multi-user jobs

Overview of multi-user job migration ...................................... 78
Application phase management .............................................. 78
Copying contents between central repositories ....................... 79
Central repository migration ............................................... 80

Index 83
Overview
Migration overview

One of the most powerful aspects of Data Integrator is its architectural flexibility through development, test, and production environments. Data Integrator is designed to support a number of environments, including large enterprises with many developers working on multiple projects. Data Integrator supports multi-site architectures whether centralized or not.

This book discusses architectural options for implementing Data Integrator in development, test, and production environments and covers the following topics:

- Migration basics
- Preparing for migration
- Export/import
- Multi-user development
- Multi-user environment setup
- Working in a multi-user environment
- Migrating multi-user jobs

This chapter also includes the following topics:

- About this document
- Audience and assumptions
- More Data Integrator product documentation

About this document

The book contains advanced development information. Topics include:

- **Migration**
  How to move your projects to different environments
- **Multi-user development**
  How to manage a project developed by multiple users

You will find this book most useful:

- After you have learned product basics
- While planning the design, test, and production phases of your data-movement projects
- As an advanced source of information during any phase of your projects
Audience and assumptions

This and other Data Integrator product documentation assumes the following:

• You are an application developer, consultant, or database administrator working on data extraction, data warehousing, or data integration.

• You understand your source data systems, RDBMS, business intelligence, and messaging concepts.

• You understand your organization’s data needs.

• You are familiar with SQL (Structured Query Language).

• If you are interested in using this product to design real-time processing, you are familiar with:
  • DTD and XML Schema formats for XML files
  • Publishing Web Services (WSDL, HTTP, and SOAP protocols, etc.)

• You are familiar Data Integrator installation environments—Microsoft Windows or UNIX.

Further, Business Objects recommends that you review both the Data Integrator Getting Started Guide and the Data Integrator Designer Guide before using advanced concepts in this document.

More Data Integrator product documentation

Consult the Data Integrator Getting Started Guide for:

• An overview of Data Integrator products and architecture

• Data Integrator installation and configuration information

• A list of product documentation and a suggested reading path

After you install Data Integrator (with associated documentation), you can view the technical documentation from several locations. To view documentation in PDF format:

• If you accepted the default installation, select Start > Programs > Business Objects > Data Integrator > Data Integrator Documentation and select:
  • Release Notes—Opens this document, which includes known and fixed bugs, migration considerations, and last-minute documentation corrections
  • Release Summary—Opens the Release Summary PDF, which describes the latest Data Integrator features
More Data Integrator product documentation

- **Technical Manuals**—Opens a “master” PDF document that has been compiled so you can search across the Data Integrator documentation suite
- **Tutorial**—Opens the Data Integrator Tutorial PDF, which you can use for basic stand-alone training purposes

Select one of the following from the Designer’s Help menu:
- **Release Notes**
- **Release Summary**
- **Technical Manuals**
- **Tutorial**

Other links from the Designer’s Help menu include:
- **DiZone**—Opens a browser window to the DI Zone, an online resource for the Data Integrator user community
- **Knowledge Base**—Opens a browser window to Business Objects’ Technical Support Knowledge Exchange forum (access requires registration)

You can also view and download PDF documentation, including Data Integrator documentation for previous releases (including Release Summaries and Release Notes), by visiting the Business Objects documentation Web site at [http://support.businessobjects.com/documentation/](http://support.businessobjects.com/documentation/).
Migration basics
About this chapter

Migration as it relates to Data Integrator is the process of moving applications through multiple development phases into production. Data Integrator supports simple and complex application migration through all phases into production.

This chapter includes the following topics:

- Development phases
- Migration mechanisms and tools

Development phases

The ETL application development process typically involves three distinct phases:

- Design phase
- Test phase
- Production phase

You can use Data Integrator in all three phases. Because each phase might require a different repository to control environment differences, Data Integrator provides controlled mechanisms for moving objects from phase to phase.

Each phase could involve a different computer in a different environment with different security settings. For example, design and initial test may only require limited sample data and low security, while final testing may require a full emulation of the production environment including strict security.
Design phase

In this phase, you define objects and build diagrams that instruct Data Integrator in your data movement requirements. Data Integrator stores these specifications so you can reuse them or modify them as your system evolves.

Design your project with migration to testing and final production in mind. Consider these basic guidelines as you design your project:

- Construct design steps as independent, testable modules.
- Use meaningful names for each step you construct.
- Make independent modules that can be used repeatedly to handle common operations.
- Use test data that reflects all the variations in your production data.

Test phase

In this phase, you use Data Integrator to test the execution of your application. At this point, you can test for errors and trace the flow of execution without exposing production data to any risk. If you discover errors during this phase, return the application to the design phase for correction, then test the corrected application.

Testing has two parts:

- The first part includes designing the data movement using your local repository.
- The second part includes fully emulating your production environment, including data volume.

Data Integrator provides feedback through trace, error, and monitor logs during both parts of this phase.

The testing repository should emulate your production environment as closely as possible, including scheduling jobs rather than manually starting them.

Production phase

In this phase, you set up a schedule in Data Integrator to run your application as a job. Evaluate results from production runs and when necessary, return to the design phase to optimize performance and refine your target requirements.

After moving a Data Integrator application into production, monitor it in the Administrator for performance and results. During production:

- Monitor your jobs and the time it takes for them to complete.
The trace and monitoring logs provide information about each job as well as the work flows and data flows contained within the job.

You can customize the log details. However, the more information you request in the logs, the longer the job runs. Balance job run-time against the information necessary to analyze job performance.

- Check the accuracy of your data.

To enhance or correct your jobs:
- Make changes in your design environment.
- Repeat the object testing.
- Move changed objects back into production.

### Migration mechanisms and tools

Data Integrator provides two migration mechanisms:

- **Export/import migration** works best with small to medium-sized projects where a small number of developers work on somewhat independent Data Integrator applications through all phases of development.

- **Multi-user development** works best in larger projects where two or more developers or multiple teams are working on interdependent parts of Data Integrator applications through all phases of development.

Regardless of which migration mechanism you choose, Business Objects recommends you prepare for migration using one or more tools that best fit your development environment (see Chapter 3: Preparing for migration for more information). The mechanism and tools you use will depend on the needs of your development environment.

If your source data will come from multiple, homogeneous systems, Business Objects recommends you use Datastore and system configurations tools. For more information on setting up datastore and system configurations, see “Creating and managing multiple datastore configurations” on page 113 of the *Data Integrator Designer Guide*. When migrating applications in a multi-user environment, Business Objects strongly recommends you use Naming conventions for migration.
Which mechanism is best?

Although Data Integrator supports a multi-user environment, you may not need to implement this architecture on all projects. If your project is small to medium in size and only consists of one or two developers, then a Central Repository may not be a necessary solution to integrating the work of those developers.

For example, only two consultants worked on a certain HR datamart application. The Development system was architected so that while Consultant 1 managed the Master Repository, Consultant 2 worked on a new section within a complete copy of the Master Repository.

Consultant 2 then exported this new section back into the Master Repository using the export utility that allows objects to be ‘Created’, ‘Replaced’, or ‘Ignored’. After updating the Master Repository, Consultant 2 took a new complete copy of the Master Repository, overwriting the previous copy.

Use the following matrix to help you determine which mechanism and tools would work best in your environment.

<table>
<thead>
<tr>
<th>Situation/requirements</th>
<th>Migration Mechanisms</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Export/import</td>
<td>Multi-user</td>
</tr>
<tr>
<td>Small to medium-sized project</td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td>Multiple-team project</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Source data from multiple, homogeneous systems</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Different source or target database among environments</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Need a “fast and easy” migration solution</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Optimal solution: X Compatible solution: O</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Export/import migration

Export/import is the basic mechanism for migrating Data Integrator applications between phases. First, you export jobs from the local repository to another local repository or to an intermediate file which you can then import.
Migration basics

Migration mechanisms and tools

into another local repository. For example, when moving from design repository to test repository, you export from the design repository to a file, then import the file to your test repository.

If you find application errors during testing, you can correct them in the development environment, then export the corrected version and import it back into the test repository for retesting. For more information on exporting, see Chapter 4: Export/import.

Multi-user migration

You can also migrate Data Integrator applications between phases in more complex development environments. Instead of exporting and importing applications, multi-user development provides a more secure check-in, check-out, and get mechanism, using a central repository to store the master copies of your application elements. Multi-user development includes other advanced features like labeling and filtering to provide you more flexibility and control in managing application objects. To learn more about multi-user development migration, see Chapter 9: Migrating multi-user jobs.
Preparing for migration
Preparing for migration

About this chapter

Before you develop Data Integrator applications, Business Objects recommends that you first set up a comprehensive structure to facilitate the migration process between development phases.

This chapter discusses tools that can help you build your migration structure:

• Naming conventions for migration
• Datastore and system configurations
• Command line login to the Designer

Business Objects recommends that you implement standardized naming conventions for connectivity between computer systems. Add datastore and system configurations to more easily work with multiple homogeneous systems.

Naming conventions for migration

The best way to ensure fast and seamless migration is to use common naming conventions across all systems and phases of all your development environments.

Just as Business Objects recommends you standardize object prefixes, suffixes, and path name identifiers to simplify your projects internally, we also recommend the use of naming conventions externally for migration purposes.

Note: To learn more about object identification in jobs, see “Naming conventions for objects in jobs” on page 74 of the Data Integrator Designer Guide.

To ease migration, use common naming conventions for:

• Connections to external datastores
• Directory locations
• Schema structures and owners

You want to make it as quick and easy as possible to migrate applications between users and between phases. This translates to significantly reducing or eliminating time spent reconfiguring your jobs to work in each specific environment.

While the actual data you are extracting, transforming, and loading usually differs by database, the essential structure of the data should be the same on every database with which you want the same applications to work. Therefore, it makes the most sense to standardize your database naming and structuring before starting the development process.
Connections to external datastores

In Data Integrator, migration is the process of moving objects between local repositories, whether directly using the Export/import method or indirectly using the Multi-user development method. Regardless of method, you must consider how the migration will impact connection configurations associated with your jobs.

Using generic naming for similar external datastore connections reduces the time you spend on reconfiguring the connections to the same database type. For example, you should choose the same logical name for all your Oracle datastore connections to the same type of database structure regardless of migration phase environment.

You can make connection names meaningful to a certain phase and specific computer system names (Test_DW, Dev_DW, Prod_DW), however if you choose this naming structure, Business Objects recommends that you use datastore configurations for migration purposes.

<table>
<thead>
<tr>
<th>Development phase</th>
<th>Test phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name: Dev_DW</td>
<td>User name: Test_DW</td>
</tr>
<tr>
<td>Password: Dev_DW</td>
<td>Password: Test_DW</td>
</tr>
<tr>
<td>Host String: Dev_DW</td>
<td>Host String: Test_DW</td>
</tr>
</tbody>
</table>

For a job to run against Test and Development, it would have to use Test_DW and Dev_DW and this would require you to create different datastore configurations for when the job runs against the Test or the Dev instance, respectively.

Alternatively, you could call the connection string DW and regardless of what instance you ran the job against, it would run without users having to create multiple datastore configurations.

<table>
<thead>
<tr>
<th>Development Phase</th>
<th>Test Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database A</td>
<td>Database B</td>
</tr>
<tr>
<td>Datastore Connection</td>
<td>Datastore Connection</td>
</tr>
<tr>
<td>User name: DW</td>
<td>User name: DW</td>
</tr>
<tr>
<td>Password: DW</td>
<td>Password: DW</td>
</tr>
<tr>
<td>Host string: DW</td>
<td>Owner name: DW</td>
</tr>
<tr>
<td>Owner name: DW</td>
<td>Owner name: DW</td>
</tr>
</tbody>
</table>

Examples:

- There is one Oracle source system in your company that processes order entry data. Multiple instances of this system exist for development, test, and production purposes. Therefore, you name the connection string to
Preparing for migration

Naming conventions for migration

your Oracle source system “ORDER_SYSTEM”. Then in all phases, you configure that name to point to the correct (phase-specific) instance of the system.

• Name the connection string to your target data warehouse “DW” then point it to different databases depending on whether you are in the development, test, or production environment.

When you use this generic, cross-phase naming method, you cannot access both dev and test from the same computer (since the connection string maps only to one instance). If you require access to both, use multiple datastore configurations.

Directory locations

Business Objects recommends you use logical directory names (for example, X:\) or point to common local drives to standardize directory location. For example, since every computer has a C drive, pointing to the directory location, C:\TEMP would be a safe, reproducible standard.

Schema structures and owners

To further facilitate a seamless structure between development phases, give all your database instances the same owner name for the same schema structures from which you are reading and to which you are loading. Regardless of name, the owner of each schema structure can vary and Data Integrator will reconcile them.
Datastore and system configurations

Datastore and system configurations are powerful tools for reducing the configurations required to execute the same logic against different datastore environments. With configurations, migration between development phases becomes faster and more simplified.

This chapter includes the following sections:

- Datastore configurations and migration
- Multiple configurations in multi-user environments
Datastore configurations and migration

Without multiple configuration datastores, each time you export/import from one repository to another, you may need to spend time reconfiguring datastore connections to work with the new repository (and sometimes new host computer).

Without multiple configurations, each job in a repository can only run against one datastore configuration.

With multiple configurations, instead of a separate datastore (and datastore configuration) for each database instance, you can associate multiple datastore configurations with a single datastore definition. For more information, see “Creating and managing multiple datastore configurations” on page 113 of the Data Integrator Designer Guide.

Each system configuration defines a set of datastore configurations that you want to use together when running a job. You must create datastore configurations for the datastores in your repository before you can create system configurations.
All objects you want to import into a multiple configurations datastore must share the same owner.
Preparing for migration
Datastore and system configurations

For specific instructions to define datastore and system configurations, see “Creating and managing multiple datastore configurations” on page 113 of the Data Integrator Designer Guide.

Multiple configurations in multi-user environments

Data Integrator also supports a multi-user development environment. A team can work together on an application during development, testing, and production phases. Further, different teams can work on the different phases simultaneously.

Individual users work on an application in their unique local repositories. The team uses a central repository to store, check in, and check out objects that belong to the application master copy. The central repository preserves all versions of an application’s objects, allowing you to revert to a previous version if needed.

The easiest way to set up your environment to work with multi-user functionality is by establishing the exact same environment naming standards among your developers. In each developer’s environment, the configuration
Preparing for migration

Command line login to the Designer

would be different. For example a database connection string would point to their local database. However, if implementing these naming standards is not possible, you can still save time and streamline your multi-user environment by using multiple-configuration datastores.

For example, if your developers use databases with the same metadata structure but different database instances and owners, you can define a datastore configuration for each developer on your design team, mapping different owners to a common set of aliases used by all. This way, they can share and contribute to the same projects without having to set up their datastore connection information each time they check out a project from the central repository.

To find out more about multi-user environments, see Chapter 5: Multi-user development, Chapter 6: Multi-user environment setup, and Chapter 8: Working in a multi-user environment. For more details on migrating within multi-user environments, see Chapter 9: Migrating multi-user jobs.

Command line login to the Designer

You can login to the Designer from the command line. This feature facilitates logging into multiple repositories, such as DEV, TEST, PROD, or different departments, and different versions.

With this feature you can create different shortcuts on your Desktop with different connection parameters, and you can choose the repository you want to connect to merely by clicking its shortcut.

The following table describes the AL_Designer command options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-U</td>
<td>User name</td>
</tr>
<tr>
<td>-P</td>
<td>Password</td>
</tr>
</tbody>
</table>
| -S      | This option contains one of the following:  
  - Database connection name for Oracle  
  - DB2 data source for DB2  
  - Database server name for SQL Server  
  - Database server name for Sybase  
  - ODBC DSN for MySQL |
| -Q      |  
  - Database name for SQL Server,  
  - Database name for Sybase |
Preparing for migration

Command line login to the Designer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-NdBTYPE</td>
<td>Database type, which can be Oracle, DB2, MySQL, Microsoft, and Sybase</td>
</tr>
<tr>
<td>-g</td>
<td>Windows authentication mode, which applies to SQL Server only</td>
</tr>
</tbody>
</table>
Export/import
Overview of export/import

The simplest type of migration in Data Integrator is called export/import.
This chapter discusses the export/import method and includes the following topics:
• Exporting/importing objects in Data Integrator
• Removing obsolete repository contents
• Backing up repositories
• Maintaining Job Server performance

Exporting/importing objects in Data Integrator

The export feature gives you the flexibility to manage and migrate projects involving multiple developers and different execution environments. When you export a job from a development repository to a production repository, you can change the properties of objects being exported to match your production environment.

In particular, you can change datastore definitions—application and database locations and login information—to reflect production sources and targets.

You can export objects to another repository or a flat file (.atl or .xml). If the destination is another repository, you must be able to connect to and have write permission for that repository, and your repository versions must match.

This section discusses:
• The Export editor
• Exporting objects to another repository
• Exporting objects to a file
• Exporting a repository to a file
• Importing from a file
• Command line options to export objects to an XML file

The Export editor

In the Export editor, specify the objects you want to export and an export location. Choose Tools > Export or select an object and right-click Export to open the export editor.

To specify an object to export, drag the object from the object library into the Objects to Export window.
The Object to Export window shows the final list of objects to be exported. When you drag any object from the object library, the datastores, file formats, and custom functions included in the object definition are automatically added to the other export sections. Each object in an export window opens to show objects called by this object.

You can control which associated objects to exclude or include. For example, you can export a work flow and all tables contained in the work flow without exporting an associated data flow.

To control which objects to export, either select an object, right-click, and choose a shortcut menu option, or select the white space in the Export editor, right-click, and choose a shortcut menu option:

- **Export**
  Starts the export process.

- **Exclude**
  Removes only the selected object from the list of objects to be exported. The object remains in the list, but its exclusion is indicated by a red “x” on the object icon.

  All occurrences of the object are excluded.

  When you export the list, excluded objects are not copied to the destination. Objects called by this object are not removed from the list of objects to be exported, unless they are specifically excluded.

- **Include**
  Adds an excluded object to the export plan. The red “X” on the icon disappears. All occurrences of the object are included.

  When you export, the included objects are copied to the destination.
• **Exclude Tree**
  Removes the selected object and all objects called by this object from the export. The objects remain in the list, but their exclusion is indicated by a red “x” on the icons—the selected object and any objects it calls are excluded.

  When you export the list, the excluded objects are not copied to the destination.

• **Include Tree**
  Add the selected excluded object and the objects it calls to the export list. The red x on the selected object and dependents disappears. When you export the list, the included objects are copied to the destination.

• **Exclude environmental information**
  Removes all connections (datastores and formats) and their dependent content (tables, files, functions) from the objects in the Export editor. Note that if you exclude datastores during export, data flows that depend on those datastores will not execute properly unless your destination repository has the same set of datastores with the same database types and versions (connection strings can be different).

  When you export, excluded objects are not copied to the destination.

  From the white space in the Export editor, right-click to select **Exclude environmental information** from the menu. Using this option you can export jobs without connections as a way to avoid connection errors. If you decide to use this option, Business Objects recommends that you configure datastores and formats for the new environment separately.

  **Note:** Business Objects recommends that instead of excluding environmental information you simply add additional datastore configurations that match the destination environment either before or after the export (and import, if exported to a file). To learn more about creating datastore configurations, see “Database datastores” on page 79 and “Creating and managing multiple datastore configurations” on page 113 of the Data Integrator Designer Guide. Also see “Datastore” on page 50 of the Data Integrator Reference Guide.

• **Clear All**
  Removes all objects from all sections of the editor.

• **Delete**
  Removes the selected object and objects it calls from the Export editor. Only the selected occurrence is deleted; if any of the effected objects appear in another place in the export plan, the objects are still exported.
This option is available only at the top level. You cannot delete other objects; you can only exclude them.

### Exporting objects to another repository

You can export objects from the current repository to another repository. However, the other repository must be the same version as the current one. The export process allows you to change environment-specific information defined in datastores and file formats to match the new environment.

1. **To export a repository object to another repository**
   1. In the object library, choose an object to export.
      - Right-click and choose **Export**.
      - The Export editor opens in the workspace. To add more objects to the list of objects to export, drag the objects from the object library into the Objects to Export section of the editor.
   2. Refine the list of objects to export.
      - You can use the options available in the right-click menu for each object to include or exclude the object from the export list.
   3. When your list is complete, right-click and choose **Export**.
   4. In the Export Destination window, add the destination database connection information.
   5. In Export Confirmation window, verify the components to export.
      - The Destination status column shows the status of the component in the target database and the proposed action.

<table>
<thead>
<tr>
<th>Destination Status</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does not exist</td>
<td>Create/Exclude</td>
</tr>
<tr>
<td>Exists</td>
<td>Replace/Exclude</td>
</tr>
</tbody>
</table>

To edit an action, select any number of objects (using SHIFT and CTRL keys) and select either **Create**, **Exclude**, or **Replace** from the Target Status list box.

6. Click **Next**.
7. In the Datastore Export Options window, select the datastore.
   - You can change the owner of a table or the connection properties of the datastore.
   - Click **Advanced**.
8. Change the database connection information as required by the target database.
   Click Next.

9. In the File Format Mapping dialog, select a file and change the Destination Root Path if necessary.
   You can change the Destination Root Path for any file formats to match the new destination.

10. Click Finish.
    Data Integrator copies objects in the Export editor to the target destination. When copying is complete, the objects display in the Output window. The Output window shows the number of objects exported as well as a list of any errors.

Exporting objects to a file

You can also export objects to a file. If you choose a file as the export destination, Data Integrator does not provide options to change environment-specific information.

Note: Objects in a repository are exported in the .atl format, while whole repositories can be exported in either the .atl or .xml format. ATL is Data Integrator’s scripting language format. Using the .xml file format might make repository content easier for you to read. It also allows you to export Data Integrator to other products.
Exporting a repository to a file

You can also export an entire repository to a file. When you export or import a repository, jobs and their schedules (created in Data Integrator) are automatically exported or imported as well. Schedules cannot be exported or imported without an associated job and its repository.

If you choose a file as the export destination, Data Integrator does not provide options to change environment-specific information.

To export a repository to a file
1. From the object library, right-click and choose Repository > Export To File.
   
   A window opens to prompt you for the destination of the export file. You can browse the directory to change the location, set the file type (.xml or .atl), and enter a name for the file.
2. Click Open.
   The repository is exported to the file.

Importing from a file

Importing objects or an entire repository from a file overwrites existing objects with the same names in the destination repository.

To import a repository from a file
1. There are two ways to import Data Integrator repository files into another repository. Use Tools > Import from file, or in the object library, right-click and choose Repository > Import from File.
   A window opens for you to specify the file to import. You can import individual files or the whole repository using either an XML or ATL file type. (ATL is Data Integrator’s internal scripting language.)
2. Select a file to import and click Open.
   • If you attempt to import an ATL file saved from an earlier version of Data Integrator, a warning displays indicating that the version of the ATL file is lower than the repository version and that the ATL file you are about to import might contain objects that do not make optimal use of your upgraded repository. For example, new options for some features might not be available. To update an ATL file, import it into a repository of the same version then upgrade that repository. To abort the import, click No. To continue with the import, click Yes.
• If you attempt to import an ATL file saved from a repository that is later than your current version, an error message displays indicating that the version of the ATL file is higher than the repository version and cannot be imported. Click OK.

3. Choose Programs > Business Objects > Data Integrator > Data Integrator Designer from the Start menu.

4. Log in to the repository where the file was imported.

5. Verify that the file or repository was imported.

Command line options to export objects to an XML file

Data Integrator provides options on the al_engine command to export an entire repository or individual objects to an XML file. This capability provides the following benefits:

• Allows external version management tools to obtain objects from the Data Integrator repository. This ability is useful if you use one version control system and want to integrate with Data Integrator.

• Facilitates automate migration from one repository to another (for example, DEV, TEST, and PROD) when you include the command in scripts.

The following table describes the al_engine command options to export to an XML file.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX</td>
<td>Full repository export. This option exports the repository to file with the name export.xml.</td>
</tr>
<tr>
<td>-XX ObjectType@FileName</td>
<td>Export all objects of the specific object type to the specified file. ObjectType can be one of the values listed in the following option.</td>
</tr>
</tbody>
</table>

The following table describes the al_engine command options to export to an XML file.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX</td>
<td>Full repository export. This option exports the repository to file with the name export.xml.</td>
</tr>
<tr>
<td>-XX ObjectType@FileName</td>
<td>Export all objects of the specific object type to the specified file. ObjectType can be one of the values listed in the following option.</td>
</tr>
</tbody>
</table>
Removing obsolete repository contents

Data Integrator saves a version of each object every time you save the object. Repeatedly modified object definitions can consume a substantial amount of space. If you notice your repository performance degrading, consider compacting the repository.

To access the Compact Repository command, select Project > Compact Repository from the menu bar. This command removes previous object versions maintained by Data Integrator.

You can also compact your repository manually. If you have never compacted the repository, the majority of space in the repository could be occupied by old versions of Data Integrator objects. In this case, the Compact Repository

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX ObjectType@FileName@Object@Name</td>
<td>Export the specific ObjectName of the specified object type to the specified file. ObjectType can be one of the following values: • P - Exports all projects • J - Exports all jobs • W - Exports all work flows • D - Exports all data flows • T - Exports all user-defined transforms • F - Exports all user-defined file formats • X - Exports all XML and DTD message formats • S - Exports all datastores • C - Exports all custom functions • p - Exports all system configurations</td>
</tr>
<tr>
<td>-XX ObjectType@FileName@Object@Name@DE</td>
<td>Export the specific ObjectName and its dependents with datastore information to the specified file.</td>
</tr>
<tr>
<td>-XX ObjectType@FileName@Object@Name@D</td>
<td>Export the specific ObjectName and its dependents without datastore information to the specified file.</td>
</tr>
</tbody>
</table>
command might be too slow and tedious. Instead, you can export the latest versions of the repository object definitions to a file, clear the repository database by creating a new repository, then reimport the object definitions.

To compact your repository by creating a new repository

1. Export the repository to a file.
   The file type can be either XML or ATL. The latest version of each object is exported.
2. Choose Repository Manager from the Start > Programs > Data Integrator menu.
3. From the Repository Manager, add the database connection information for the repository.
4. Click Create.
   Data Integrator warns that a valid repository already exists.
5. Click Yes to overwrite the old repository.
   The Repository Manager creates a new repository, removing all of the old objects.
6. Import the previously exported repository.

Backing up repositories

Use your DBMS utilities to back up your repositories regularly. For information, refer to your DBMS documentation.

Maintaining Job Server performance

If you are designing jobs, typically you might use the same computer for your Designer, repository, and Job Server. In addition, you might use the same datastore for both your repository and your target database.

However, when you migrate your jobs into a test environment, the Job Server could move to a separate computer (typically from a Windows to a UNIX platform). The Data Integrator Job Server computer uses source, target, and repository database client libraries to extract, transform, and load data according to a job’s design. Therefore, the Job Server computer must have a database client installed for each database you are using to run a Data Integrator job. In addition, Data Integrator allows you to localize source and target databases using locale and codepage settings. For more information about locales, see “Locales and Multi-Byte Functionality” on page 645 of the Data Integrator Reference Guide.
When migrating jobs between different Job Servers verify that the codepage used by each source and target database is the same as the codepage set for the corresponding database client on the Job Server’s computer.

The database client codepage used by a Job Server on a Windows might be different from the one used on UNIX. For example, the Oracle client codepage MS1252 on Windows should be changed to the ISO88591 codepage on UNIX.

Data Integrator allows different codepages to be used in sources and targets. Mismatched locale settings do not cause errors and Data Integrator attempts to treat equivalent settings without any transcoding. However, mismatches may result in performance degradation from transcoding done by Data Integrator during job execution.

If your jobs do not require the use of different locales, you can increase performance by ensuring that default locales are not mismatched. After migration, if you notice a significant difference between the speed of design and test environments, check locale settings. In the Designer, check to see that datastore codepages for sources and targets match client codepages on the Job Server computer.
About multiple users

Data Integrator supports a multi-user development environment. A team can work together on an application during the development, testing, or production phase. Also, different teams can work on the different phases at the same time.

Each individual developer works on an application in their unique local repository. Each team uses a central repository to store the master copy of its application. The central repository preserves all versions of all objects in the application so you can revert to a previous version if necessary.

This chapter discusses:
• Central versus local repository
• Data Integrator and multiple users

Central versus local repository

Data Integrator allows you to create a central repository for storing the team copy of a Data Integrator application. The central repository contains all information normally found in a local repository such as definitions for each object in an application. However, the central repository is merely a storage location for this information. To change the information, you must work in a local repository.

A local repository provides a view of the central repository. You can “get” (copy) objects from the central repository into your local repository. However, to make changes to an object, you must “check out” that object from the central repository into your local repository. While you have an object checked out from the central repository, other users cannot check out that object, so they cannot change the information.

After completing changes, you “check in” the changed object. When you check in objects, Data Integrator saves the new, modified objects in the central repository.

Multiple users working from unique local repositories can connect to the same central repository. These users can work on the same application and share their work. However, at any given time only one user can check out and
change a particular object. While an object is checked out to one user, other users can "get" (obtain a copy of) the object but cannot make changes that will update the central repository. The central repository retains history for each object. Therefore, if you find you made a change that did not work as planned, you can revert to a previous version of the object.

The local repository and the central repository must use the same Data Integrator repository version. For example, you can run Data Integrator Designer X.2 with a central and local repository version X.1. However, you cannot run Data Integrator Designer X.2 with a central repository X.1 and a local repository X.2

Data Integrator and multiple users

A multi-user environment affects how you use Data Integrator and how you manage different phases of an application. For success in a multi-user environment, you must maintain consistency between your local repository and the central repository.

The following terms apply when discussing multi-user environments and Data Integrator:

• Highest level object

  The highest level object is the object that is not a dependent of any object in the object hierarchy. For example, if Job 1 is comprised of Work Flow 1 and Data Flow 1, then Job 1 is the highest level object.
• Object dependents
  Object dependents are objects associated beneath the highest level object in the hierarchy. For example, if Job 1 is comprised of Work Flow 1 which contains Data Flow 1, then both Work Flow 1 and Data Flow 1 are dependents of Job 1. Further, Data Flow 1 is a dependent of Work Flow 1.

• Object version
  An object version is an instance of an object. Each time a you add or check in an object to the central repository, Data Integrator creates a new version of the object. The latest version of an object is the last or most recent version created.

When working in a multi-user environment, you activate the link between your local repository and the corresponding central repository each time you log in. To ensure that your repository is current, you can get (copy) the latest version of each object in the central repository. Once you get an application in your local repository, you can view and run it from the Designer.

However, if you plan to make changes to objects in the application, you must check out those objects. After you check out an object, no other user can make changes. Essentially, you lock the version in the central repository; only you can change that version. Other users can only get and view the object.
When you are done making changes to an object, save those changes in the local repository and check the object back into the central repository. Data Integrator saves the changed object in the central repository and makes the object available for check-out by others. Data Integrator maintains all versions of saved objects in the central repository. Thus later, you can copy an old version of a saved object, even after replacing it in your local repository with a new version.

At any time, you can label an object or a group of objects. An object label provides a convenient mechanism for identifying objects later. For example, you may find it helpful to label objects by feature. Later, if you decide you want to eliminate a recently-added feature, you can get all objects that have the label without that feature.

You can also compare two objects—such as two different object versions in the central repository, or an object in your local repository to an object in the central repository. By comparing two objects, you can determine what parts of an object changed and decide whether you want to revert to an older version of an object. For more information on object comparisons, see “Comparing Objects” on page 437 of the Data Integrator Designer Guide.

Security and the central repository

Data Integrator also provides options to make your central repository secure. Use these options when you need to control access and provide for object tracking within your central repository. These security options apply only to central repositories and include:

- Authentication — Allows only valid users to log in to a central repository.
Multi-user development

Security and the central repository

- Authorization — Grants various levels of permissions to objects.
- Auditing — Maintains a history of changes made to an object including user names.

Implement security for a central repository by establishing a structure of groups and associated users using the Administrator. For more information about how to implement security for your central repository, see Chapter 7: Implementing Central Repository Security.
Multi-user environment setup
Overview of multi user setup

To support multiple Data Integrator developers, configure a multi-user environment and set up several repositories. Specifically, you must:

• Create a local repository for each developer
• Create a central repository—see “Create a nonsecure central repository”.
• Define a connection to central repository from each local repository—see “Define a connection to a nonsecure central repository” on page 47.
• Activate the connection to a central repository—see “Activating a central repository” on page 47.

Create a nonsecure central repository

To support multiple Data Integrator users in a single development environment, Business Objects recommends that you use a central repository. The central repository stores master information for the development environment.

This procedure applies to nonsecure repositories only. To create a secure central repository, see Chapter 7: Implementing Central Repository Security.

► To create a nonsecure central repository
1. Create a database to be used for the central repository using your database management system.
2. From the Start menu, choose Programs > Data Integrator > Repository Manager (assuming you installed Data Integrator in the Data Integrator directory).
3. In the Repository Manager window, click the Central button in the Repository Type field, and enter the database connection information for the central repository.
4. Click Create.
   Data Integrator creates repository tables in the database you identified.
Define a connection to a nonsecure central repository

A team working on an application only needs one central repository. However, each team member requires a local repository. Furthermore, each local repository requires connection information to any central repository it must access.

This procedure applies to nonsecure repositories only. To define the connection to a secure central repository, see Chapter 7: Implementing Central Repository Security.

**Note:** The version of the central repository must match the version of the local repository.

1. Start the Data Integrator Designer and log in to your local repository.
2. Choose **Tools > Central Repositories** to open the Options window. The **Central Repository Connections** option is selected in the **Designer Options** list.
3. Right-click in the **Central Repository Connections** box and select **Add**. The Datastore Administrator window opens.
4. In the **Name** box, enter a name to identify your central repository.
5. In the **Database Type** list, select the appropriate database type for your central repository.
6. Complete the appropriate login information for your database type.
7. Click **OK**.

The list of central repository datastores now includes the newly connected central repository. You can continue adding additional connections or you can proceed to the next step.

Activating a central repository

To connect to a central repository, you must activate the link between your local repository and a specific central repository.

**Note:** When you start the Designer, always log in to a local repository. Never log into a central repository. If you do, then the central repository acts as a local repository. Then you run the risk of corrupting version information. If you attempt to log in to the central repository, Data Integrator will present a warning message. You should log out immediately and log into a local repository.
Your local repository provides a view of the objects in the active central repository. Whenever you get or check out objects, you copy objects from the active central repository. Whenever you check in objects, you save the version from your local repository into the active central repository.

You must activate the correct central repository each time you log in. When you activate a central repository, Data Integrator opens the central object library, which shows all the objects in the central repository and the check-out status of each object.

► **To activate a central repository**

1. Choose **Tools > Central Repositories** to open the Options window.
   - The **Central Repository Connections** option is selected in the **Designer Options** list.
2. In the **Central repository connections** list, determine a central repository to make active.
3. Check **Reactivate automatically** if you want the active central repository to be reactivated when you next log on to this local repository.
4. Right-click the central repository and select **Activate**.
   - Data Integrator activates the link between your local repository and the selected central repository and opens the central object library. The Options window indicates that the selected central repository is active and closes automatically.
To open the central object library

Click the **Central Object Library** button on the toolbar.

The central object library looks like the object library—it shows all the objects in the repository, grouped on appropriate tabs.

The window opens in floating mode. Drag the window to dock it. To change the docking state, right-click the Central Object Library tool bar and toggle **Allow Docking**.

You can also change central repository connection information from the central object library.

To change the active central repository

Select a central repository from the list on the top of the central object library. Data Integrator makes the selected central repository active—objects from that repository appear in the central object library. Connection information about that repository appears in the upper right corner of the central object library.

To change central repository connections

1. Click the **Edit Central Repository Connection** button on the top of the central object library.
Data Integrator opens the Options window with the Central Repository Connections option selected in the Designer Options list. Alternatively, you can open the Options window by selecting Tools > Central Repositories.

2. Select a central repository in the Central Repository Connections box, right-click, and select Edit.

The Datastore Administrator window opens.

Follow the instructions beginning at step 4 to define a connection to the central repository. See “Activating a central repository” on page 47.

3. To delete connection information for a central repository, right-click the central repository in the Central Repository Datastores box and select Delete.

After confirming your selection, Data Integrator deletes the connection information from this local repository. You can no longer connect to that central repository from this local repository.

Note: You are not deleting the central repository; you are only deleting the connection information between your local repository and this central repository.

4. To make another repository the active central repository, right-click the central repository in the Central Repository Datastores box and select Activate.

5. To disconnect from the currently active central repository, right-click the central repository in the Central Repository Datastores box and select Deactivate.
Implementing Central Repository Security
About this chapter

This chapter describes how to implement optional security features for central repositories and includes the following sections:

- Overview
- Creating a secure central repository
- Adding a multi-user administrator (optional)
- Defining a connection to a secure central repository
- Working with objects in a secure central repository

Overview

Data Integrator provides options for managing secure access and tracking for objects in central repositories. Mechanisms for managing central repository security include:

- Authentication — Allows only valid users to log in to a central repository.
- Authorization — Grants various levels of permissions to objects.
- Auditing — Maintains a history of changes made to an object including user names.

Note that these security mechanisms and procedures apply only to central repositories.

Group-based permissions

You implement security for a central repository by establishing a structure of groups and associated users using the Administrator. (You can optionally add a user to the Administrator with the role Multi-user administrator. For details, see “Managing user roles” on page 29 of the Data Integrator Management Console: Administrator Guide.)

Access permissions for objects apply at the group level. More than one group can have the same permissions to the same object at a time. Groups are specific to a repository and are not visible in any other local or central repository.

Therefore, users do not get individual permissions. In the Designer, users select from the group(s) to which they belong, and the selected (current) group dictates their access to that object. Each user must have one default
group but can belong to more than one group. When a user adds an object to a secure central repository, the user’s current group automatically has Full permissions to that object.

User name and password authentication is required for every logon to a secure central repository. Users can change their passwords at any time from the Central Repository Editor in the Designer.

**Permission levels**

Each object in a secure central repository can have one of the following permissions levels:

- **Full** — This is the highest level of permission. The group can perform all possible actions including checking in, checking out, and deleting the object. You might assign this type of access to developers, for example.
- **Read** — Users can only get a copy of the object from the central repository or compare objects between their local and central object libraries. You might assign this type of access to QA, for example.
- **None** — Users cannot get copies of the object but can view it and its properties.

When an authenticated user adds an object to a secure central repository, the user’s current group receives Full permissions to the object. All other groups receive Read permissions. Members of the group with Full permissions can change the other groups’ permissions for that object.

**Process summary**

You implement security for a central repository by:

1. **Creating a secure central repository**—Use the Repository Manager to add a secure central repository or upgrade an existing nonsecure central repository.
2. **Adding a multi-user administrator (optional) and Setting up groups and users**—Use the Administrator to add groups and users.
3. **Defining a connection to a secure central repository**—Define the connection from the Designer.
4. **Working with objects in a secure central repository**—Add objects to the central repository as well as view and modify object permissions.
Creating a secure central repository

The first step in establishing security measures for multi-user development is to create a secure central repository or upgrade an existing nonsecure central repository.

Note: These procedures apply to secure repositories only. To create a nonsecure central repository, see Chapter 6: Multi-user environment setup.

To create a secure central repository
1. Create a database to be used for the central repository using your database management system.
2. From the Start menu, click Programs > BusinessObjects Data Integrator > Repository Manager (assuming you installed Data Integrator in the Data Integrator directory).
3. In the Repository Manager window, click the Central button in the Repository Type field and enter the database connection information for the central repository.
4. Select the Enable security check box.
5. Click Create.

Data Integrator creates repository tables in the database you identified.

Data Integrator creates a security key file with a name in the form of databaseserver_database_user.key.

To upgrade a central repository from nonsecure to secure
You can modify an existing central repository to make it secure; however, you cannot undo this change.

1. From the Start menu, click Programs > BusinessObjects Data Integrator > Repository Manager (assuming you installed Data Integrator in the Data Integrator directory).
2. In the Repository Manager window, click the Central button in the Repository Type field and enter the database connection information for the central repository to modify.
3. Select the Enable security check box.
4. Click Upgrade.
   Data Integrator updates the repository tables in the database you identified.
   Data Integrator creates a security key file with a name in the form of databaseserver_database_user.key.

**Adding a multi-user administrator (optional)**

In the Administrator, you have the option of adding a user with the role of Multi-user Administrator. This role is limited to managing secure central repositories, so it is therefore a subset of the Administrator role. For example, Multi-user Administrators cannot add a local repository or a nonsecure central repository.

Multi-user Administrators can:

- Add and remove secure central repositories.
- Manage users and groups.
- View secure central repository reports.

For details, see “Managing user roles” on page 29 of the Data Integrator Management Console: Administrator Guide.

**Setting up groups and users**

The next step in implementing central repository security is to add and configure groups and users with the Data Integrator Administrator.

For detailed procedures, see “Managing user roles” on page 29 of the Data Integrator Management Console: Administrator Guide.

**Defining a connection to a secure central repository**

The next step in implementing central repository security is to define a connection to the repository in the Designer.

This procedure applies to secure central repositories only. To define a connection to a nonsecure central repository, see Chapter 6: Multi-user environment setup.
Implementing Central Repository Security

Defining a connection to a secure central repository

To define a connection to a secure central repository

1. Start the Data Integrator Designer and log in to your local repository.

2. From the **Tools** menu, click **Central Repositories** to open the Options window.

   The **Central Repository Connections** option should be selected in the **Designer** list.

3. Click **Add** to open the Central Repository Editor.

4. In the **Repository Name** box, enter a name to identify the connection to this central repository (this name is only visible in the Options window in the Central Repository Connections list).

5. Click the **Secure** check box.

6. This enables the **Repository User Information** fields and the **Security** key buttons.

7. In the **Database Connection Information** area, your options are:
   
   a. Click **Read Security Key** to import the database connection information values from the key that Data Integrator generated when you created the secure central repository using the Repository Manager.

   OR

   b. Manually enter the appropriate database and user information.

   (Optional) To save this information for future use, click **Generate Security Key**, which creates a key file with a name in the form of `databaseserver_database_user.key`

8. In the **Repository User Information** area, enter the user name and password as defined in the Administrator.

9. (Optional) Click the **Remember** check box to store this information for the next time you log in.

10. (Optional) Click **Change Password** to change the user’s password for accessing this secure central repository. In the Change Password dialog box, type the current password, the new password, then again to confirm it, and click **OK**.

11. Click **OK**.

   The list of central repository connections now includes the newly connected central repository and it is identified as being secure.

12. With the repository selected, click **Activate**.

   For more information about activating central repositories, see “Activating a central repository” on page 47.

13. Click **OK**.
Working with objects in a secure central repository

This section includes:
• Adding objects to the central repository
• Viewing and modifying permissions

Adding objects to the central repository

See “Adding objects to the central repository” on page 61.

Viewing and modifying permissions

After completing all configuration tasks and adding objects to the secure central repository, use the central object library to view and modify group permissions for objects.

To view permissions for an object
1. Start the Data Integrator Designer and log in to your local repository.
2. Open the secure central object library.

Your default group appears in the drop-down list at the top of the window and is marked with an asterisk. The Permissions column displays the current group’s access level for each object. If you add a new object to the central library, the current group gets FULL permissions and all other groups get READ permission.
Implementing Central Repository Security

Working with objects in a secure central repository

- **To change object permissions to other groups**
  You must have Full permissions to change object access to other groups.
  1. In the central object library, right-click the object and click **Permission > Object** or **Permission > Object and dependants**.
  2. The **Permission** dialog box opens, which displays a list of available groups and the group’s access level for the object(s).
  3. Click in the **Permission** column, and from the drop-down list select a permission level for the group.
  4. Click **Apply** or **OK**.

- **To change the current group or the default group**
  1. To change the current group, in the central object library select a group from the drop-down box.
  2. To change your default group, select the desired group from the drop-down box and click the save icon.
    Data Integrator marks the default group with an asterisk.
Working in a multi-user environment
Overview of multi-user tasks

To obtain optimal results from development in a multi-user environment, Business Objects recommends certain processes, such as checking in and checking out objects that you change, and establishing a set of conventions that your team follows, such as labeling objects.

You will complete several tasks on a regular basis:

- Filtering
- Adding objects to the central repository
- Checking out objects
- Undoing check out
- Checking in objects
- Labeling objects
- Getting objects
- Comparing objects
- Viewing object history
- Deleting objects

Filtering

Data Integrator allows you to customize by filtering (selectively changing) environment-specific information in object definitions. Application objects can contain repository-specific information. For example, datastores and database tables might refer to a particular database connection unique to a user or a phase of development. When multiple users work on an application, they can change repository-specific information.

Specifically, filtering allows you to:

- Change datastore and database connection information
- Change the root directory for files associated with a particular file format
- Select or clear specific dependent objects

The filtering process is available when adding, checking in, checking out, or getting latest objects in a central repository.

When you select any command that uses the filtering option, Data Integrator presents the following two windows:
1. The Version Control Confirmation window shows your selected object and any dependent objects. You can exclude objects by selecting the object and changing the **Target status** (lower right) from *create* or *replace* to *exclude*.

![Version Control Confirmation](image)

2. The Datastore Options window shows any datastores used by the object. This window only opens if the objects that you are adding, checking in, or checking out include a datastore.

![Datastore Options](image)

### Adding objects to the central repository

After creating a central repository, connecting it to the local repository, and activating the central repository, you can add objects from the local repository to the central repository. Remember that you do all design work—the creation of jobs, work flows, and data flows—in a local repository. Therefore, you use a local repository for the initial creation of any objects in an application. After the
initial creation of an object, you add it to the central repository. Once in the central repository, the object is subject to version control and can be shared among users.

You can add objects to the central repository at any point. Of course, you cannot add an object that already exists in the central repository.

You can add a single object to the central repository, or you can add an object with all of its dependents to the central repository. When you add a single object, such as a data flow, you add only that object. No dependent objects are added.

To add a single object to the central repository
1. Open the local object library.
2. Right-click the object and select Add to Central Repository > Object.
3. The Comments window opens. Enter any comments in the Comments field, and click OK.

   Data Integrator adds the object to the active central repository.

   Note: The Add to Central Repository command is not available if the object already exists in the central repository.

To add an object and its dependent objects to the central repository
1. Open the local object library.
2. Right-click the object and select either Add to Central Repository > Object and dependents or Add to Central Repository > With filtering (if filtering is required).
3. The Comments window opens. Enter any comments in the Comments field, and click OK.
4. If you selected With filtering, complete the filtering windows, as described in “Filtering” on page 60.
5. Click Finish to add the selected objects.

Alternatively, you can select the object and drag it to the central object library to add the object and its dependents to the central repository. The filtering windows are displayed.

Note: The Add to Central Repository command is not available if the object already exists in the central repository. However, the Add to Central Repository command is available if the object’s dependents already exist in the central repository but the object itself does not.
Checking out objects

When you might change any of the objects in an application, you should check out the objects that you expect to change. When you check out an object, you make that object unavailable to other users—other users can view the object but cannot make changes to the object. Checking out an object ensures that two users do not make conflicting changes to the object simultaneously.

Data Integrator changes the object icons in both the local and central object libraries to indicate that the object is checked out.

![Object icons: checked out vs. not checked out]

When an object is checked out, your central object library shows you the local repository that has checked out the object. Based on the repository name, you can determine which user is working with that object.

To see periodic changes, refresh the central object library by clicking on the refresh central object library icon in the toolbar of the central object library.

Choose a check-out command based on what you will do to an object:
- Check out single objects or objects with dependents
- Check out single objects or objects with dependents without replacement
- Check out objects with filtering

Check out single objects or objects with dependents

Dependents are objects used by another object—for example, data flows that are called from within a work flow. You can check out a single object or an object with all of its dependents (as calculated in the central repository). For example, you can simply check out a work flow. In that case, you can change that work flow, such as adding a new script to the work flow; however, you cannot change dependent objects in the work flow, such as data flows, and retain the changes in the central repository. Changes to dependent objects will only be retained in the local repository. Alternatively, you can check out...
the work flow with all of its dependents. In that case, you can make changes to the work flow or any of its dependents and retain the changes in both central and local repositories.

Generally, it is safest to check out an object with all dependents. When you do this, you prevent others from accidentally changing dependent objects.

▶ To check out a single object
1. Open the central object library.
2. Right-click the object you want to check out.
3. Choose Check Out > Object.

Alternatively, you can select the object in the central object library, and click the Check Out Object button on the top of the central object library.

Data Integrator copies the most recent version of the selected object from the central repository to your local repository, then marks the object as checked out.

▶ To check out an object and its dependent objects
1. Open the central object library.
2. Right-click the object you want to check out.
3. Choose Check Out > Object and dependents.

Alternatively, you can select the object in the central object library, and click the Check out object and dependents button on the top of the central object library.

Data Integrator copies the most recent version of the selected object and all of its dependent objects from the central repository and marks these objects as checked out.
If a dependent object is checked out by you or another user, then Data Integrator alerts you with a Check Out Alert window, asking to get the latest version of the checked out object. See “Getting objects” on page 71.

Check out single objects or objects with dependents without replacement

When you check out an object, you can replace the object in your local repository with the latest version from the central repository, or you can leave the current version in your local repository intact.

When you check out an object, Data Integrator copies the object definition from the central repository and replaces any existing definitions for that object in your local repository.

You can check out objects without replacing the objects in your local repository. For example, suppose you are working in your local repository and you make a change to an object that is not checked out. If you determine that the change improves the design or performance of your application, you will want to include that change in the central repository.

To do this, check out the object without replacing the object in your local repository—the object that you have already improved with a change. Then, check the changed object back into the central repository.

Note: Use caution when checking out objects without replacing the version in your local repository. When you do not replace the version in your local repository, you can lose changes that others have incorporated into those objects.

To check out an object or an object and its dependent objects without replacement

1. Open the central object library.
2. Right-click the object you want to check out and choose Check Out > Object without replacement to check out the single object or choose Check Out > Object and dependents without replacement to check out the object and all of its dependent objects.

Data Integrator marks all appropriate objects as checked out—in both the object library and in the workspace—but does not copy any objects from the central repository to the local repository.
Check out objects with filtering

When you check out an object with filtering, the object and all its dependents are checked out.

**Note:** When you check out objects with filtering, you always replace local versions with the filtered objects from the central repository.

1. Open the central object library.
2. Right-click the object you want to check out and choose **Check Out > With filtering**.
3. Complete the filtering windows, as described in “Filtering” on page 60.
4. Click **Finish** to check out the selected objects.

Undoing check out

Occasionally, you may decide that you did not need to check out an object because you made no changes. Or, you may decide that the changes you made to a checked-out object are not useful and you prefer to leave the master copy of the object as is. In these cases, you can undo the check out.

When you undo a check out:

- the object in the central repository remains as it was before the checkout; no changes are made and no additional version is saved in the central repository. Only the object status changes from checked out to available.
- the local version of the object maintains the changes you made. If you want the local object to be an exact copy of the central object, perform a **Get latest** operation on that object.

After you undo a check out, other users can check out and make changes to the object.

1. Open the central object library.
2. Select a checked-out object.
3. Click the **Undo object check out** button.
Data Integrator removes the check-out symbol and makes no changes to the object in the central repository. Any checked-out dependent objects remain checked out.

Alternatively, you can right-click the object and select **Undo Check Out > Object**.

**To undo check out of an object and its dependents**

1. Open the central object library.
2. Select the checked-out object that is the highest level for which you want to undo the check out.
3. Click the **Undo object and dependents check out** button.

Data Integrator removes the check-out symbols for the object and any dependent objects that are also checked out. No changes are made to these objects in the central repository.

Alternatively, you can right-click the object in the central object library and select **Undo Check Out > Object and dependents**.

### Checking in objects

After you finish making changes to checked out objects, you must check them back into the central repository. Checking in objects creates a new version in the central repository, and allows others to get the changes that you have made. Checking in objects also preserves a copy of the changes for revision control purposes. Later, you can get a particular version of a checked in object and compare it to subsequent changes or even revert to the previous version.

Check in an object when you are done making changes, when others need the object that contains your changes, or when you want to preserve a copy of the object in its present state.

Choose a check-in command based on what you will do to an object:

- Checking in single objects, objects with dependents
- Checking in an object with filtering
Checking in single objects, objects with dependents

Just as you can check out a single object or an object with all dependent objects, you can check in a single object or an object with all checked-out dependent objects (as calculated in the local repository).

▶ To check in a single object
1. Open the central object library.
2. Select the object you want to check in.
3. Click Check in object button at the top of the central object library.

Alternatively, you can right-click the object in the central object library and select Check In > Object.

4. A Check In window opens with a Comment box, in which you can enter comments. After entering any comments, click OK.
Data Integrator copies the object from your local repository to the central repository, and removes the check-out mark.

▶ To check in an object and its dependent objects
1. Open the central object library.
2. Select the highest level object you want to check in.
3. Click Check in object and dependents button at the top of the central object library.

Alternatively, you can right-click the object in the central object library and select Check In > Object and dependents.

4. A Check In window opens with a Comment box, in which you can enter comments. After entering any comments, click OK.
Data Integrator copies the selected object and all of its dependent objects from your repository to the central repository and removes the check-out mark.
Checking in an object with filtering

Just as you could check out objects with filtering, you can check in objects with filtering. When you check in an object with filtering, the object and all its dependent objects are checked in.

To check in an object with filtering
1. Open the central object library.
2. Right-click the object you want to check out and choose Check In > With filtering.
3. A Check In window opens with a Comment box, in which you can enter comments. After entering any comments, click OK.
   Data Integrator warns you that you are about to create a new version of the object in the central repository.
4. Click Yes to continue with the check in.
5. Complete the filtering windows, as described in “Filtering” on page 60.
6. Click Finish to check in the selected objects.

Labeling objects

To help organize and track the status of objects in your application, you can label objects. When an object is labeled, the object and all its dependent objects are labeled. A label not only describes an object, but also allows you to maintain relationships between various versions of objects.

For example, suppose developer A adds a job to the central repository and works on a work flow in that job while developer B works on a data flow in the same job. At the end of the week, after developer A checks in two versions of the work flow and developer B checks in four versions of the data flow to the central repository, the job is labeled “End of week 1 status.” This label contains version 1 of the job, version 2 of the work flow, and version 4 of the data flow. Both developers can continue to change their respective work flow and data flow.
At some later point, if you want to get the job with the version of the data flow with this label, getting the job by its label (see “Getting objects” on page 71) accomplishes this, whereas checking out the job and its dependents does not.

The label “End of week 1 status” serves the purpose of collecting the versions of the work flow and data flow that were checked in at the end of the week. Without this label, you would have to get a particular version of each object in order to reassemble the collection of objects labeled “End of week 1 status.”

1. Open the central object library.
2. Right-click the object you want to label and choose Label. Data Integrator opens the Get by Label window.
3. In the Label box, enter text that describes the current status of the object, then click OK.

Data Integrator inserts this label in the history of the object and its dependents. To view the labels associated with a particular object, see “Viewing object history” on page 72.

Getting objects

To make sure that your repository is up-to-date, you “get” objects. When you get an object, you copy the latest version of that object in the central object library and copy it into your local repository, replacing the version in your local repository. When you get an object, you do not check out the object. The object remains free for others to check out and change.

You can get an object with or without dependent objects and filtering. For information about getting earlier versions of objects or objects with particular labels, see “Viewing object history” on page 72.

To get a single object
1. Open the central object library.
2. Select the object you want to get.
3. Click Get latest version of object at the top of the central object library.

Alternatively, right-click the object in the central object library and select Get Latest Version > Object.
Data Integrator copies the most recent version of the object in the central repository to your local repository.

To get an object and its dependent objects
1. Open the central object library.
2. Select the highest level object you want to get.
3. Click Get latest version of objects and dependents at the top of the central object library.

Alternatively, right-click the object in the central object library and select Get Latest Version > Object and dependents.
Working in a multi-user environment

Comparing objects

Data Integrator copies the most recent version of the selected object and all dependent objects from the central repository to your local repository.

To get an object and its dependent objects with filtering
1. Open the central object library.
2. Select the highest level object you want to get.
3. Right-click the object in the central object library and select Get Latest Version > With filtering.
4. Complete the filtering windows, as described in “Filtering” on page 60.
5. Click Finish to get the selected objects.

To get a previous version of an object
See “To get a previous version of an object” on page 72.

To get an object with a particular label
See “To get an object with a particular label” on page 72.

Comparing objects

Data Integrator allows you to compare two objects from local and central repositories to determine the differences between those objects.

For details on using the Difference Viewer, see “Comparing Objects” on page 437 of the Data Integrator Designer Guide.

Viewing object history

The central repository retains a history of all changes made to objects in the central repository. Use this history to help manage and control development of your application. For example, you can use the central repository:

- To examine the history of an object
- To get a previous version of an object
- To get an object with a particular label

To examine the history of an object
1. Open the central object library.
2. Select an object.
3. Click the Show History button at the top of the central object library.
Alternatively, you can right-click the object in the central object library and select **Show History**.

Data Integrator opens the History window.

This window shows several pieces of information about each revision of the object.

<table>
<thead>
<tr>
<th><strong>Column</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>The object revision number. Each time a user saves the object, Data Integrator creates a new version.</td>
</tr>
<tr>
<td>Label</td>
<td>Text that a user enters to describe the status of the object at a given point. See “Labeling objects” on page 69 for more information about object labeling.</td>
</tr>
<tr>
<td>Repository</td>
<td>Information about the local repository from which Data Integrator saved this version of the object. Other information includes:</td>
</tr>
<tr>
<td></td>
<td>• user name</td>
</tr>
<tr>
<td></td>
<td>• database connection name</td>
</tr>
<tr>
<td></td>
<td>• type of database</td>
</tr>
<tr>
<td>Date</td>
<td>The date and time Data Integrator saved this version of the object.</td>
</tr>
</tbody>
</table>
Working in a multi-user environment

Deleting objects

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The type of change a user made to the object. This table records actions such as:</td>
</tr>
<tr>
<td></td>
<td>Checked in — User checked in object</td>
</tr>
<tr>
<td>Comment</td>
<td>Comments a user enters when adding an object or checking it into a central repository.</td>
</tr>
</tbody>
</table>

To get a previous version of an object
1. Select an object.
2. Click the Show History button at the top of the central object library.
3. Click the version of the object you want.
4. Click the Get Obj By Version button.

Note: When you get a previous version of an object, you only get the object but not its dependent objects.

To get an object with a particular label
1. Select an object.
2. Click the Show History button at the top of the central object library.
3. Click the version of the object with the particular label you want.
4. Click the Get By Label button.

Deleting objects

You can delete objects from either the central repository or a local repository. To delete an object from the central repository, right-click the object in the central object library and select Delete. To delete an object from the local repository, right-click on the object in the object library and select Delete.

When you delete an object from a local repository, you do not automatically delete that object from the active central repository. In fact, you can get the object from the central repository to re-insert it.

Similarly, when you delete an object from a central repository, you do not automatically delete the object from connected local repositories. Until you delete the object from the local repository, you can add the object back to the central repository.

When you delete objects from the central repository, you only delete the selected object and all versions of the selected object; you do not delete any dependent objects.
To delete all versions except the latest version of an object from the central repository, use the compact repository. Select **Project > Compact Repository** from the menu bar.
Working in a multi-user environment

*Deleting objects*
Migrating multi-user jobs
Overview of multi-user job migration

Job migration applies to Data Integrator on multiple levels: application level, repository management level, and product upgrade level. Application migration is much more flexible in a multi-user environment, allowing you to maintain not only multiple versions of your objects during development, but also during test and production phases if you choose.

- Application phase management
- Copying contents between central repositories
- Central repository migration

Application phase management

Typically, applications pass through different phases on the way from development to production. For example, an application might pass through three phases:

- Developers creating an application
- Testers validating the application
- Administrators running the application

A single central repository can support your application through all phases. Use job labeling and projects to maintain application components independently for each phase. For example, if development wants to make a certain version of an application ready for testing, they may label it "APPL_V1". Testers can then get that particular application version using the label and proceed with testing. If testing is successful, an administrator can get the application to run in the production environment. In addition, datastore configurations and file locations allows you to configure the application to run in each local environment.

In some situations, you may require more than one central repository for application phase management. If you choose to support multiple central repositories, use a single local repository as a staging location for the transition.

In some situations, you may require more than one central repository for application phase management. Following the example above, once developers create an application version ready for testing by labeling it, a tester would get that version from the development central repository, test it and then check it into a test central repository.
That test central repository will contain all versions tested over time, allowing flexibility for testers to go back to any previous version without relying on the development environment. When an application version passes testing, an administrator can get it from the test repository and make it available in production. Again, if you need to maintain previous versions of an application already in production, you can create another central repository.

With this scheme, a developer will never interfere with the test environment, and a tester will never interfere with a production environment, creating an extremely safe process of migration.

Note that if you choose to support multiple central repositories, use a single local repository as a staging location for file transition.

Copying contents between central repositories

You cannot directly copy the contents of one central repository to another central repository. Rather, you must use your local repository as an intermediate repository.

- To copy the contents of one central repository to another central repository
  1. Activate the central repository whose contents you will copy. See “Activating a central repository” on page 47.
2. Get the latest version of all objects in this active central repository so they exist in your local repository. See “Getting objects” on page 71.

3. Activate the central repository into which you want to copy the contents. See “Activating a central repository” on page 47.

4. The first time you copy the contents, add the objects from your local repository into this central repository. See “Adding objects to the central repository” on page 61.

   However, if you must re-copy the contents of one central repository into another (for example, during your testing phase some part of a job was reassigned to the development phase for redesign), the process is slightly more complex:

   a. First check out specific objects without replacement from the second central repository. See “Checking out objects” on page 63.

   b. From your local repository, get the latest version of the objects from the first (for example, development) central repository.

   c. Then, instead of adding, check in the updated objects from your local repository to the second (for example, test) central repository. See “Checking in objects” on page 67.

**Central repository migration**

When you upgrade your version of Data Integrator, you should migrate your central repository to the new version. Business Objects recommends that consider the following guidelines when migrating a central repository to a new release of Data Integrator.

1. Back up all central repository (as well as local repository) database tables and associated data before upgrading.

2. Maintain a separate central repository for each version of Data Integrator to preserve object history.

   To preserve the current version and history of objects in your central repository, create a new central repository of your current version of Data Integrator and copy the contents of the original central repository to the newly-created one (see Copying contents between central repositories). This way, the second central repository acts as a backup for your objects and associated history information from the older version of Data Integrator.

   When you install the new version of Data Integrator, upgrade the newly-created central repository to the latest version of Data Integrator.
3. Coordinate efforts to upgrade your central repositories and local repositories at the same time.

   Different versions of your central and local repository may not work together. You cannot perform a multi-user operation between a local and central repository of different Data Integrator version.

4. Check in all objects (or undo check-outs if objects were not modified after they were checked out) before migrating the central repositories.

   If you cannot upgrade your central and local repositories at the same time, you should check in all objects (or undo check-outs if objects were not modified during check-out), especially those objects checked out to a local repository you will not be immediately upgrading. After you upgrade your central repository to the new version of Data Integrator, you will not be able to check in objects from the local repository of the older version of Data Integrator.
9 Migrating multi-user jobs
Central repository migration
Index

C
central object library 49
central repositories
activating 48
adding a single object 62
adding objects 61
adding objects and dependents 62
changing 49
connection to 47
copying contents of 79
creating 46
multiple 78
overview 40
checking in
object and its dependent objects 68
object with filtering 69, 72
single object 68
checking out
object and its dependent objects 64
object and its dependent objects with filtering 66
object and its dependent objects without replacement 65
object without replacement 65
single object 64
undoing, object and its dependent objects 67
undoing, single object 66
Compact Repository command 35

designing ELT projects 13
developing applications
design phase 13
production phase 13
testing phase 13

E
Export editor options 28–31
exporting
command line options 34
objects 28–32
repository to a file 33–33
repository versions 31

F
filtering
checking in object with 69, 72
checking out objects with 66
description 60–62

G
generic naming 19
getting
object and its dependent objects 71
object with label 74
objects 71
previous version of object 74
single object 71

H
highest level object 41
history of objects 72

J
job server
performance, matching locales 36
L
label
  getting object with 74
object 69, 70
locales, job server performance 36
logical directory names 20

M
migration, naming conventions 18
multi-user development
  application development 41
  environment, setting up 46
  overview 40
  repository versions 41
tasks 60–??

N
names, logical directory 20
naming conventions for migration 18
naming datastore connections 19

O
objects
  adding to central repository 62, 62
  checking out 63
deleting 74
dependents of 42
getting 71
highest level 41
history 72
label 69, 70
version 42

P
performance
  job servers, matching locales 36
production phase 13
profiles, datastore and system 18

R
repository
  export, to .xml or .atl 32
  exporting 33
  importing 33
  removing obsolete contents from 35
versions 31, 41

T
 test phase 13
 testing applications 13

U
 undo
 checking out a single object 66
 checking out object and its dependent objects 67

V
 versions
 object, deleting 74
 object, description 42
 object, getting 74
 repository 31, 41