COGNOS®

Cognos

Transformer(R)

Transformer for UNIX Guide

Product Information

This document applies to Transformer (R) Version 7.1 and may also apply to subsequent releases. To check for newer versions of this document, visit the Cognos support Web site (http://support.cognos.com).

Copyright

Table of Contents

Welcome

What Is in This Book

This guide contains the information you need to use the PowerPlay Transformer UNIX Edition:

- a description of Transformer in a client-server environment
- an explanation of how to set up client-server communications
- the procedure for creating server models and cubes
- information about managing the Transformer on UNIX environment with user preference files and environment variables
- details about how to run Transformer on UNIX from the command line
- guidelines for production and maintenance

What You Need to Know to Use This Book Effectively

To use this book effectively you should

- first read the *PowerPlay Administrator's Guide*
- know how to design and build a PowerPlay system
- know the UNIX operating system

Other Documentation

An annotated list of other documentation, the *Documentation Roadmap*, is available from the Windows Start menu or the Transformer Help menu.Our documentation includes user guides, tutorial guides, reference books, and other pieces to meet the needs of our varied audience.

All information is available in online help. Online help is available from the Help button in a Web browser, or the Help menu and Help button in Windows products.

The information in each online help system is available in online book format (PDF). However, the information from a given help system may be divided into more than one online book. Use online books when you want a printed version of a document or when you want to search the whole document. You can print selected pages, a section, or the whole book. Cognos grants you a non-exclusive, non-transferable license to use, copy, and reproduce the copyright materials, in printed or electronic format, solely for the purpose of providing internal training on, operating, and maintaining the Cognos software.

In Windows products, online books are available from the Windows Start menu (Cognos) and from the product Help menu (Books for Printing). In a Web browser, online books may be available from the Welcome section of the help system, or from within the Cognos Web portal (Upfront). All online books are available on the Cognos documentation CD. You can also read the product readme files and the installation guides directly from the Cognos product CDs.

Only the installation guides are available as printed documents.

An annotated list of other documentation, the *Documentation Roadmap*, is available from the Windows Start menu or the Transformer for UNIX Help menu.

Questions or Comments?

For the fastest response to questions about using PowerPlay Transformer, contact customer support.

For information about customer support locations and programs, see Cognos on the Web on the Help menu or visit the Cognos Support Web site (http://support.cognos.com).

Chapter 1: Overview of Transformer on UNIX

This chapter provides an overview of how Transformer works in a client-server environment.

The topics covered are

- Two modes of Transformer on UNIX
- Why Use Transformer on UNIX
- Components of Transformer on UNIX
- Transformer on UNIX and Cubes
- Design Server Models and Cubes
- The Client-Server Communications Process

Two Modes of Transformer on UNIX

Transformer on UNIX has two modes:

- Command line (rsserver) on UNIX

  For information about running UNIX from the command line, see "Run Transformer on UNIX from the Command Line" (p. 8).
- Client-Server Transformer or Windows using PowerGrid

  For information about using PowerGrid, see "Communications" (p. 7).

Why Use Transformer on UNIX?

Transformer on UNIX offers additional capabilities to those available with Transformer on Windows, including

- increased performance. Depending on the size of the server, Transformer on UNIX may be able to generate models and cubes faster than local processing on a computer.
- easier integration of cube builds and data warehouse updates. It is easier to integrate these processes if they both run under the same operating system.
- faster processing. If your database resides on the same UNIX server, processing performance and build times may improve as a result of a reduction in network traffic.

Components of Transformer on UNIX

PowerPlay Transformer UNIX Edition includes components such as communication software, model prototyping and synchronization, Server menu commands, and configuration, which are available exclusively with this edition. You can also run Transformer on UNIX from the command line.

Communications

Transformer on Windows communicates with Transformer on UNIX via PowerGrid, a utility that is included with PowerPlay Transformer UNIX Edition. The PowerGrid network daemon (udanetd) must be installed, configured, and running on the computer where Transformer on UNIX listens for requests from Transformer on Windows. All communications from Transformer on Windows to PowerGrid must be made using a Windows Sockets compliant TCP/IP connection.

For information about client-server communications, see "Define a Client-Server Connection" (p. 12).

Model Prototyping and Synchronization

Each server model can be derived from a local model you create using Transformer on Windows. Each local model is associated with one server version of the same model. The two versions of Transformer (Transformer on Windows and Transformer on UNIX) ensure that these models remain synchronized with one another.

For information about model prototyping, see the section "Prototype a Model" (p. 15). For information about synchronization, see "Synchronize Models" (p. 19).

Server Menu Commands

Many of the actions you can perform locally using Transformer on Windows, such as generating categories and creating cubes, you can also perform on the server using Transformer on UNIX. The Server menu in Transformer on Windows contains commands for setting up client-server connections and for creating models and cubes on the server.

For information about these commands, see "Define a Client-Server Connection" (p. 12) and "Create Server Models and Cubes" (p. 15).

Server Configuration

Various server preference settings control how Transformer on UNIX runs.

For information about the server environment configuration, see "Manage the Transformer on UNIX Environment" (p. 23).

Run Transformer on UNIX from the Command Line

You can run Transformer on UNIX from the command line (independent of Transformer on Windows).

- For information about the command line syntax, see "Run Transformer on UNIX from the Command Line" (p. 33).

Transformer on UNIX and Cubes

You can use Transformer on UNIX to create cubes that are binary compatible with the ones you create in the Windows environment. Once they are created, you have the option to copy them to a Windows-based LAN environment or let users access them directly from the server via a file-access facility such as NFS or Samba.

For more information, consult your UNIX administrator.

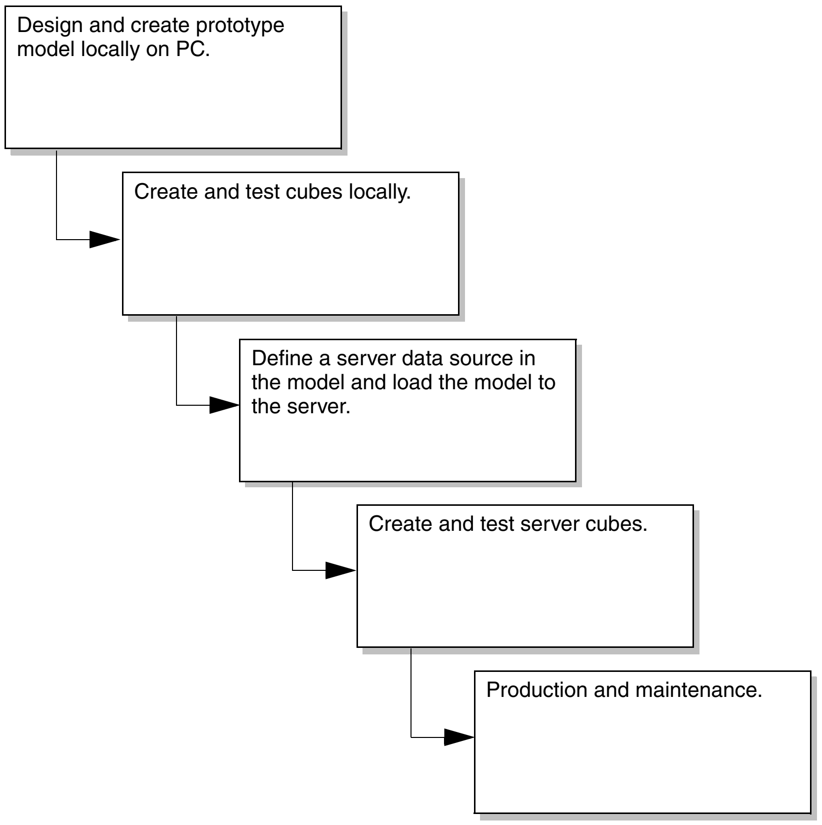
When IQD files are used, it is essential that connectivity exists between the UNIX computer and the database. Any database that is available through your database connectivity software can be accessed using Transformer on UNIX.

Design Server Models and Cubes

The development of a server model and associated cubes typically begins on Windows. Using Transformer on Windows and local data sources that are structurally identical to the ones you will use for your server model, you build and test a prototype model and create cubes.

Once you have determined that the local model and cubes are appropriate, you define a server data source in the model and store the model on the server. With a server model in place, you can start Transformer on UNIX (rsserver) from Transformer on Windows or from the command line to generate categories for the server model and create cubes using server data sources.

Test the server models and cubes to verify that they are providing the required information. Once this is complete, you can set up a regular production environment for the ongoing creation of cubes and their deployment to users. The following diagram shows the model design and cube creation process.


Design and create prototype model locally on PC.
Create and test cubes locally.
Define a server data source in the model and load the model to the server.
Create and test server cubes.
Production and maintenance.

The Client-Server Communications Process

When you use the Server menu to issue commands from Transformer on Windows to Transformer on UNIX, Transformer on Windows and Transformer on UNIX communicate according to the following process:

1. Using a Windows Sockets compliant TCP/IP connection, Transformer on Windows passes the request to the network daemon (udanetd), the PowerGrid network daemon.
2. The network daemon receives the request, starts Transformer on UNIX (rsserver), and connects it to Transformer on Windows. After that, Transformer on Windows and rsserver communicate directly with each other via TCP/IP and library routines. The rsserver program gets information in the form of Model Definition Language (MDL) verb statements from Transformer on Windows. The rsserver program uses settings defined in the environment where PowerGrid was started and searches for the Transformer preferences files (trnsfrmr.rc and .trnsfrmr.rc) that provide additional operational settings.
3. Once started, Transformer on UNIX carries out the requests, such as generating categories for a model or creating cubes, on the server.

An important part of this communication process is synchronization.

For information about synchronization, see "Synchronize Models" (p. 19).

Once the server has completed a request, it remains running for a pre-set period of time, awaiting other requests. The udanetd remains available indefinitely to handle new requests.

Chapter 2: Set Up Client-Server Transformer Communications

This chapter describes communications between Transformer on Windows and Transformer on UNIX.

The topics covered are

- PowerGrid
- Define a Client-Server Connection
- Transformer on UNIX Shell Scripts

PowerGrid

When you issue commands from Transformer on Windows to Transformer on UNIX, these commands are transmitted by PowerGrid. PowerGrid is a dedicated utility that is installed on the server along with Transformer on UNIX. The role of this utility is to provide a Windows interface to Transformer on UNIX, which enables you to control Transformer on UNIX from within Transformer on Windows. The PowerGrid network daemon (udanetd) listens for requests from Transformer on Windows to initiate a Transformer on UNIX task.

For information about setting up Transformer on UNIX and PowerGrid, see the *Installation Guide for PowerPlay Transformer Edition (UNIX)*.

Before you can create server models and cubes, you must ensure that

- both PowerGrid udanetd and Transformer on UNIX are installed and configured on the computer where you will be creating server cubes.
- a local connection is defined on the Windows computer for the server where udanetd and Transformer on UNIX are installed.

  For information about how to define a connection, see "Define a Client-Server Connection" (p. 12).
- the server environment is set up so that Transformer on UNIX creates models and cubes where and how you want them created.

  For information about the server environment, see "Manage the Transformer on UNIX Environment" (p. 23).

For information about the steps to configure a client-server environment, see "Configuration Checklist for a Client-Server Environment" (p. 43).

While you would normally use PowerGrid to start Transformer on UNIX, you can directly logon to UNIX and issue commands to Transformer on UNIX.

For information about how to issue commands to Transformer on UNIX, see "Run Transformer on UNIX from the Command Line" (p. 33).

How PowerGrid Works

Following is a description of how PowerGrid sets up communications between Transformer on Windows and Transformer on UNIX:

1. PowerGrid udanetd is running on the server, listening for messages on the pre-configured IP port. The default is 1526. You can change the default either from the Configuration Manager command processor (configure), or by editing the PowerGrid start-up script (netpgrc) directly.
2. From Transformer on Windows, you issue a server command, for example, from the Server menu, Maintenance submenu, Restore Server Model from Client.

   This sends a message to the port to run the rsserver.sh script. The default location of this script is cer*n*\bin.

   Note: *n* is the Cognos rendition number (one digit). For example: cer3.

   For information about the rsserver.sh script, see "Transformer on UNIX Shell Scripts" (p. 13).
3. PowerGrid hears the message and runs rsserver.sh, passing the Transformer on Windows address to the shell script.
4. The rsserver.sh script starts rsserver, the Transformer on UNIX executable.
5. Transformer on UNIX sends a message to Transformer on Windows to confirm that it has started.

For the remainder of the session, Transformer on Windows communicates directly with Transformer on UNIX, without using PowerGrid.

Define a Client-Server Connection

You can create and modify connection definitions by using

- the Maintenance submenu of the Server menu in Transformer on Windows
- the Server tab in the Model property sheet (from the File menu, click Model Properties) in Transformer on Windows
- the NetInfo utility, which is described in the following sections

For information about defining connections on the server, see the Transformer online Help.

The commands on the Maintenance submenu are not available until you have actually created a new model. Before you attempt to communicate with Transformer on UNIX, it is good practice to use NetInfo to test any connections that you have defined.

Add a Connection Using NetInfo

NetInfo is a utility that enables you to add and test connections to PowerGrid on the server.

Steps to Create a Connection

1. Start NetInfo.
2. From the File menu, click Edit Connections.
3. Click New to add a new connection.
4. In the Network Type box, click Windows Sockets TCP/IP.

   The entries shown depend on what is defined in your cogconnect.ini file. By default, cogconnect.ini contains an entry for Windows Sockets. However, it may contain additional entries if you are using other Cognos products.
5. In the Host Name box, click or type the name of the server on which PowerGrid and Transformer on UNIX are installed.

   If NetInfo is able to find a Hosts file, it will provide you with a list of host names from that file. If no Hosts file is found, you must type the name or the IP address of the server on which PowerGrid and Transformer on UNIX are installed. If you don't know the name of the server, contact your network administrator.
6. In the Host Type box, click UNIX.

7. In the User Name box, type your login ID for the computer specified in the Host Name box.

   Transformer uses this ID when connecting to the server.

   Note: Under certain circumstances, you must click the Setup button to change the local configuration for the connection you are creating. If the network port ID was not set to 1526 when PowerGrid was configured, you must change your local port setting to match the network port ID. If it was set to 1526, you can skip steps 8 to 10 and proceed to step 11.

8. Click Setup.

9. Click Use Specific Port for This Connection, and type a number for the network port.

   Your local port ID must match the network port. In the following example, the cer*n*/bin/netpgrc script on the server where PowerGrid is running shows the value 3125. Therefore, you would enter 3125 in the Use Specific Port for This Connection box.

   ```
   # port number
   NPNETD=3125
   ```

10. Click OK, and click Save.

11. In the Connection to be Saved box, type a name for your connection, click OK, and then click Done.

Test a Connection Using NetInfo

Once you have defined a connection, and before you issue a command to Transformer on UNIX for the first time, use NetInfo to test the connection. Using NetInfo is similar to using a ping utility when testing network connections; it verifies that messages are being sent to udanetd, and that udanetd is responding to them.

Steps to Test a Connection

1. Start NetInfo.
2. From the Network menu, click Test Connection.
3. In the Connection Name box, double-click the name of the connection you want to test.

NetInfo sends a test packet to udanetd via PowerGrid.

If the connection is working correctly, NetInfo shows a message that looks like this:

```
Network: Microsoft Windows Sockets Version 1.1.
The current address is ‘[142.80.66.251#0]’
---- Test ‘1’ ----
Reply received from udanetd owned by ‘root’  [PID=’818’]
```

If the connection is not working, verify that

- udanetd is running on the server
- the port number defined for your connection matches the one defined on the server

Transformer on UNIX Shell Scripts

When PowerGrid starts Transformer on UNIX, it does so using an entry in the cogconnect.ini file. The entry looks like this:

```
[Service - Transformer Server]
NETWORK=rsserver.sh
```

PowerGrid udanetd uses the shell script rsserver.sh to start Transformer on UNIX when a request is issued from Transformer on Windows. Before you attempt to use Transformer on UNIX, ensure that the Transformer server service is defined in your cogconnect.ini file and that the shell script is in the search path when starting udanetd on the server. You can also edit your rsserver.sh file to set up environment variables, which Transformer on UNIX uses to generate models and create cubes.

Note: If the Transformer server service is not defined in the cogconnect.ini file, run the Configuration Manager to apply Transformer configuration parameters.

For information about environment variables, see "Environment Variables" (p. 30).

Chapter 3: Create Server Models and Cubes

This chapter describes how to use Transformer on Windows to set up prototype models for Transformer on UNIX. In addition, it describes how to start rsserver from Transformer on Windows to create models and cubes on the server.

The topics covered are

- Use Transformer on UNIX via a Remote Computer
- Prototype a Model
- Set Up Data Sources for the Prototype
- Change Settings to Build Server Models and Create Server Cubes
- Synchronize Models
- Set up security on UNIX

Use Transformer on UNIX via a Remote Computer

To use Transformer on Windows to control Transformer on UNIX on a remote computer, you must

- set up a connection to the server.

  For information about setting up a connection, see "Define a Client-Server Connection" (p. 12).
- configure the server environment before you can communicate with Transformer on UNIX and create server models and cubes.

  For information about configuring the server environment, see "Manage the Transformer on UNIX Environment" (p. 23).

Prototype a Model

Server models often start with a prototype model that you create using Transformer on Windows. Once the prototype model is developed to your satisfaction, use Transformer on UNIX to create the server version of the model. Each server model is associated with one client model.

The process of prototyping a client-server model is similar to creating any other model. The difference is how you set up the appropriate data sources for the local prototype.

Set Up Data Sources for the Prototype

Models you create using Transformer on UNIX must read data sources from the server where Transformer on UNIX is running. Because Transformer on UNIX is intended for large cube production, it is likely that these data sources contain large volumes of data. When you prototype a server model on the client, you will need to set up data sources that provide access to a sufficient subset of the data source to enable you to design your model. You can use any data source type supported by Transformer when creating your prototype model.

The data sources you use to build the prototype must adhere to the following rules:

- They must have the same columns as those you intend to use later for Transformer on UNIX.
- The names of columns used to define levels and measures in the models must match.
- The columns can be in any order and may be unused.
- There may be fewer categories in each column of the local data source than in the server data source. If this is the case, data records found in the server data source, which were not found in the local data sources, will provide new categories for the server model.

Use Supported Data Sources for Server Models

Transformer on UNIX uses the following data source types:

- Impromptu Query Definition (.iqd) files
- Delimited-field text (with or without column titles)
- Fixed-field text or fixed-field and record without CR LF
- Cognos PowerHouse Subfiles

If you use data sources other than Impromptu .iqd files, you must set up separate physical sources locally on the computer for your prototyping, and on the server.

Note: In Transformer on Windows, when you are defining a data source that will be used by Transformer on UNIX, the only valid data source types are those listed above.

Use Impromptu to Set Up Data Sources

You can use Impromptu to set up a local data source for your prototype model. You can set up an .iqd file that returns a subset of the data you want to use in the model. Impromptu enables you to filter data in many ways. You can

- filter by value on specific columns. For example, you could include data for only a few of your regions when generating categories for the Regions dimension in the model.
- return a specific number of rows. For example, if the server data source contains two million records, you might want to return only the first 10,000 of them to use for model prototyping.
- return only unique records if the server data source contains a lot of duplicate records

When you use an .iqd file to prototype your model, Transformer on Windows incorporates the contents of the .iqd file into the model. Later, when you have Transformer on UNIX generate categories and create cubes, it will access the same database that you used to build the prototype.

Steps to Build Prototype and Server .iqd Data Sources

1. In Impromptu, create an .iqd file that returns sufficient data to build a prototype model.
2. In Transformer on Windows, design and build the prototype.
3. Use Impromptu to remove any limits or filters that you applied to restrict data when you defined the original .iqd file, and re-save the file.
4. In Transformer on Windows, use the Modify Columns command to match the columns in the model with those in the new .iqd file, and to load new columns into the model.
5. In Transformer on Windows, on the Data Source property sheet for each data source, change the Data Source Location to Server.

Example

In a sales analysis model, there are Impromptu reports defined for Products, Regions, and Sales Transactions. Within Impromptu, you set the limit for row retrieval to a relatively small value (500 rows, for example). You then save these reports in .iqd format, and use Transformer on Windows to read them and define the model structure. Then, using Impromptu, you remove the 500 row restriction from each report and save the .iqd files again. Within Transformer on Windows, you change each data source to a server data source. You can then generate server categories and create server cubes using the .iqd files with no restrictions set on data retrieval.

Change Settings to Build Server Models and Create Server Cubes

By default, Transformer assumes that models are being created for local processing. As a result, none of the Server menu commands for server model and cube creation are available until you change some model settings. These settings define

- how Transformer on Windows communicates with Transformer on UNIX
- how and where Transformer on UNIX builds the server model and creates cubes

For information about the steps involved in configuring a client-server environment, see "Configuration Checklist for a Client-Server Environment" (p. 43).

Steps to enable the commands in the Server menu and to send requests from Transformer on Windows to Transformer on UNIX

1. Enter server information so that Transformer on Windows can connect to the server and the server knows where to create the server model file.
2. Provide server data source definitions so the server can access data from sources on the server.
3. Provide server cube definitions so that category generation and cube creation occurs on the server.

Once you have made these changes, the commands in the Server menu are available for use.

For information about how to control the server environment, see "Manage the Transformer on UNIX Environment" (p. 23).

Enter Server Information

Steps to enter server information

1. From the File menu, click Model Properties, and then click the Server tab.
2. In the Model Path box, type the file name you want Transformer on UNIX to use when creating the server model file.

   You can include a server directory name.
3. Use the Connections box and, optionally, the Connections button, to specify the name of a connection you have set up to communicate with Transformer on UNIX.

Provide Server Data Source Definitions

Transformer on UNIX can process only server data sources. By changing the location of your data sources from local to server, you identify a location where Transformer on UNIX can access the server data required to build the server model and create server cubes.

Steps to change the data source location from local to server

1. In Transformer on Windows, in the Data Sources window, double-click the data source that will become the server data source to open the Data Source property sheet, and then click the Source tab.
2. Set the Data Source Location to Server.
3. If the Source Type is not Impromptu Query Definition, in the Server Data File box, type the name of a data source file that Transformer on UNIX can use for the model.
4. Repeat steps 1 to 3 for each data source used to create the server model.

   Ensure that, for each local data source file, there is an equivalent data file available on the server.

Note: If you are generating cubes, the file settings (local or server) for your data sources must match the file settings for your cubes (processed locally or on the server). You can use local data sources to generate cubes locally. If you are creating models and cubes using the server, the Data Sources list must contain at least one server transaction data source that is processed on the server. You can generate a model structure either on the client or the server, since model information is synchronized between the client and server.

Change the Cube Definition to Server

By providing a server cube definition in Transformer on Windows, you specify that category generation and cube creation are done by Transformer on UNIX.

Steps to change the Cube definition to server

1. Open the PowerCube property sheet, and click the Processing tab.
2. Change the Processed location to On the Server.
3. Click the Output tab and type the cube file name.
4. Repeat steps 1 to 3 for each cube you want Transformer on UNIX to process.

Create a Server Model

Once you have defined all the settings required for a server model (using the Model property sheet), you can create a server model.

Once you have created a server model, it has a permanent association with the local model from which it was created. Transformer ensures that the Local and Server versions of the model always contain the same information.

For information about synchronizing models, see "Synchronize Models" (p. 19).

Note: If you are going to copy your client model to the server manually, such as using an FTP program, and then run your model from the command line, you should save the model in ASCII (.mdl) format rather than binary (.py?) format. You can't run a Windows .py? model on the server from the command line, because Windows and UNIX py? formats are not binary compatible. (The ? in .py? is replaced by the character used in your version of Transformer.)

Step to create a server model

- From the Maintenance submenu of the Server menu, click Restore Server Model from Client.

  Transformer sends the information contained in the local model to the server, where Transformer on UNIX processes and stores the model as a server model.

Tip: To have Transformer on UNIX create cubes, you can use the Create Server Cubes command.

Note: Although we recommend that you use the commands in the Server menu to create server models, you can also use other methods. You can

- save the model as a Model Definition Language (.mdl) file and copy that file to the server. You can then run Transformer on UNIX from the command line to process the .mdl file and create a model.
- create the model on the server using MDL verb statements, either manually (this is not recommended for the novice user) or using a third party tool that generates MDL.

Generate Categories on the Server

In the Server menu, there are two commands you can use to generate server categories:

- Generate Server Categories, which uses all server data sources to generate server categories
- Generate Categories from Selected Server Data Source, which uses only the selected server data source to generate server categories

Both commands cause PowerGrid to start a Transformer on UNIX task, or connect to an active Transformer on UNIX task, which uses the defined server data sources to generate categories on the server. As part of server category generation, Transformer synchronizes the local model with the server model. As a result, when you generate server categories, new categories are reflected in your model diagrams.

Create Server Cubes

Once you have created a server model and generated categories for that model using server data sources, you can use Transformer on UNIX to create the cubes associated with that model.

In the Server menu, there are two commands you can use to create server cubes:

- Create Server PowerCubes, which creates all the server cubes defined in the cubes list
- Create Selected Server PowerCube, which creates only the selected cube

Both commands cause PowerGrid to start a Transformer on UNIX task or connect to an active Transformer on UNIX task, which creates cubes on the server.

Multiprocessing

If the model setting for multiprocessing is enabled, rsserver starts a process called rsserverda, which handles the read phase of column calculation, category generation, and cube creation. These records are then passed to Transformer on UNIX for processing. The rsserverda program closes automatically once it has read a data source. A separate instance of rsserverda is started for each data source that uses multiprocessing. If Transformer on UNIX does not complete successfully or rsserverda does not close automatically, use the kill command to close rsserverda.

The MDL setting that enables multiprocessing is MultiProcessing True, which should appear in the statement that defines a data source (usually the DataSourceMake statement).

Synchronize Models

Within a client-server implementation, every server model has a one-to-one relationship with a client model. This means that only one client version of a model can be associated with only one server version of that model. Transformer doesn't allow two or more users to work simultaneously on a server model using different versions of the associated client model.

Within each model created both locally and on the server, Transformer maintains internal numeric identifiers—cycle numbers and timestamps—that identify a client model and a server model. Transformer uses these identifiers to synchronize the contents of the client and server models that you create.

Why is Synchronization Required?

Synchronization enables Transformer on Windows to keep in step with Transformer on UNIX so you can use the Transformer Windows user interface to maintain your models. Synchronization prevents model versions from diverging when either the client or server model changes.

When production runs are completed on the server, synchronization ensures that new categories added from server data sources are incorporated into the client model. Once this is complete, you can make changes to the model using the Transformer on Windows interface.

Automatic Synchronization

When you choose a command from the Server menu, Transformer on Windows automatically synchronizes the corresponding model at the start and end of the command. Synchronization involves

- sending changes made in the client model to the server model and incorporating them there
- sending changes made in the server model back to the client model and incorporating them there

If both the client and server model have been changed independently, the changes made to the client model have precedence.

Manual Synchronization

If a server model is updated independently of Transformer on Windows, the client model becomes outdated. This can happen, for example, when you run an MDL script on the server to update a model and the script causes new categories to be added to the server model.

To keep client and server models synchronized, you need to open the model in Transformer on Windows and manually initiate synchronization. Before you change a client model, you should synchronize it, so that any changes made to the server model are reflected in the client. This ensures that the client model is current.

Step to Synchronize Models Manually

- From the Server menu, click Synchronize.

Restore Model Files

When the client and server models both exist but cannot be synchronized, Transformer shows the following message:

```
The client and server models are not synchronized.
```

This can occur, for example, when you have generated categories for the server model and either forgotten to save, or decided not to save, the client copy of the model. Synchronization fails because Transformer detects inconsistencies in the internal synchronization stamps.

When this happens, you can use the commands in the Maintenance submenu of the Server menu to update the client model with the server model or to update the server model with the client model, whichever is most suitable. If changes were made on the server (as a result of a production run that has processed new data, for example), you would update the client model based on the server model. Conversely, if you have made changes locally, you would update the server model from the client.

Example

You create a client model using Transformer on Windows and a server model from the client using the Restore Server Model from Client command. Later, you use the rsserver directly from the command line to create a server-based cube. If the server data source includes records for which no categories exist in the model, Transformer on UNIX creates new categories.

The server model is now updated; however, the client model is out of date. The next time you start Transformer on Windows, open the client model, and connect to Transformer on UNIX, Transformer compares the file contents and time stamps. Since they do not match, Transformer copies the new categories from the server model to the client model.

Automate Model Synchronization

Below is a sample MDL script that saves the model on the client after synchronizing the client and server models and creating the cubes Training, Skills Inventory, and Staff Growth:

```
Openmdl "model.mdl"
CreateFromCubes OnServer "Training" "Skills Inventory" "Staff Growth"
Savemdl "model.mdl"
```

Set Up Security on UNIX

Transformer on UNIX can call Access Manager directly so that you can update security information in the cube entirely on UNIX.

If you have defined operating system signons in Access Manager, Transformer models that contain security information will be synchronized automatically. This process is similar to security on Windows, except that you can’t use basic signons on UNIX.

Alternatively, you can use MDL verbs to update security information. With MDL verbs, you can

- create, update, or delete a user class or a user class view
- create, update, or delete items in a user class list view
- add or remove user classes from a cube

Before using MDL to create a cube, you must

- ensure that the model contains authentication information. You will need to add it if it doesn't exist.
- create required dimension views
- create user classes. Enter 0 (zero) for the Id so that rsserver will query the LDAP server
- create the user class list
- add the user classes to the cube
- save your model

Before running rsserver from the command line, you must

- have properly configured Access Manager to work with the LDAP server. For more information, see the *Configuration Manager User Guide*.
- ensure that each user who is running Transformer has an operating system signon that is defined in Access Manager and matches the user's UNIX logon name
- set the environment variables for the data sources and the cube; for more information, see "Environment Variables" (p. 30)

Once you have completed these activities, you can do one of the following:

- build your authenticated model from the command line
- use MDL verbs to add or update security information for existing models

The following example shows how to add authentication information to a model, using MDL verbs.

```
OpenMDL "modelname.mdl"
// Opens the unauthenticated model.

DataSourceUpdate "national98-2000" Userclasses True AccessManagerName "manyclass1"
AccessManagerLogin "PPuser" AccessManagerPassword "PPuser" AccessManagerUserclass
"class1"
// Sets the Access Manager namespace and supplies the login //information.

ViewMake "class1~User View" Dimension "Line" ViewUserClass "class1" Apex "Dishwashers"
// Creates user class views.

UserClassMake "Root User Class" Id 0 DimensionView "Date" "All Categories" DimensionView
"Line" "All Categories"
DimensionView "State" "All Categories" DimensionView "Market" "All Categories"
MeasureInclude "Revenue" Yes
MeasureInclude "Cost" Yes MeasureInclude "Quantity" Yes
UserClassMake "class1" Id 0 DimensionView "Date" "All Categories" DimensionView "Line"
"All Categories"
DimensionView "State" "All Categories" DimensionView "Market" "All Categories"
MeasureInclude "Revenue" Yes
MeasureInclude "Cost" Yes MeasureInclude "Quantity" Yes
// Adds user classes. Id 0 means that Access Manager will be queried for the correct IDs.

UserClassListUpdate "Root User Class" StartList "class1" EndList
UserClassListUpdate "class1" StartList EndList
// Puts the list in order.

PowerCubeUserListUpdate Cube "userclass" StartList "Root User Class" "class1" EndList
// Adds user classes to the cube.

CubeUpdate 207 MDCFile "national.mdc" ServerCube True

CreateFiles
// Builds the cube.

SaveMDL "national_modified.mdl"
// Saves the newly authenticated model.
```

If you need to update user class configuration automatically in Transformer on UNIX, you must ensure that the Access Manager UNIX environment is properly configured. Otherwise, Transformer on UNIX will build a cube with the security that was defined when the model was last saved in Transformer on Windows.

If you are using MDL, but you are not using Access Manager authentication, you need to use the -k command line option to process signons.

For more information about the -k command line option, see (p. 35).

Chapter 4: Manage the Transformer on UNIX Environment

This chapter describes the overall management of the Transformer on UNIX (rsserver) environment and gives details of the settings for the user preference files and environment variables.

The topics covered are

- Control Transformer on UNIX with Preferences and Environment Variables
- Where rsserver Obtains Settings
- How rsserver Uses Settings
- Rules for Preference File Entries
- Preference Settings
- Environment Variables

Control Transformer on UNIX with Preferences and Environment Variables

When rsserver is called from Transformer on Windows or from the command line, it populates models and creates cubes, writes messages to a log file, and performs other Transformer actions. How and where these actions are performed is determined by preferences and environment settings that you provide.

You can

- create user preference files on the host system, which control the operating characteristics of rsserver when it is started from Transformer on Windows or from the command line.
- set up UNIX environment variables.
- override a specific setting by specifying a command-line option when invoking rsserver.

Where rsserver Obtains Settings

The rsserver program searches for settings in the order shown here:

1. trnsfrmr.rc in the rsserver program directory
2. .trnsfrmr.rc in the current working directory
3. .trnsfrmr.rc in the user's home directory
4. environment variables
5. the -D or -F command-line switches when running rsserver

   Note: If the command line contains both -D and -F, rsserver uses the one that appears last.

The settings contain information such as default directories for various classes of files that rsserver uses or creates, timeout values, and communication variables.

How rsserver Uses Settings

All preference file settings can be used as environment variables. You can use multiple preference files. The rsserver program reads all the preference files it finds. However, it is important to note that settings in each successive location override settings in previous locations. This means that settings in .trnsfrmr.rc in the home directory override settings in both .trnsfrmr.rc in the current working directory and trnsfrmr.rc in the program directory. Environment variable settings override settings in any preference file, and command-line options override environment variable settings.

Example 1

To use an environment variable to override the directory where rsserver reads data source files, include an environment variable definition such as the following in the rsserver.sh file:

```
DataSourceDirectory=$HOME/data; export $HOME/data
```

Example 2

To use the command line to override the setting for the directory where rsserver reads data source files, start rsserver as follows:

```
rsserver -D DataSourceDirectory=$HOME/data
```

Rules for Preference File Entries

When reading a setting from a preference file, rsserver observes these rules:

- Entries are not case-sensitive.

  Note: Environment variables are case-sensitive.
- Blank characters, tab characters, and lines beginning with the number sign (#) are ignored.
- In most cases, rsserver ignores a line if an invalid command-line switch is specified and provides usage instructions. However, if you specify invalid values, such as the name of a file that does not exist, rsserver may write errors to the log file. For example, this will happen if the entry for ModelSaveDirectory is incorrect, but if the entry for CubeSaveDirectory is wrong, the variable will be ignored and no entry will be made to the log file.
- If the setting for any of the following preferences is changed to a value that is outside the acceptable range, the invalid setting is changed at runtime. Either the maximum or minimum value is used, depending on which value is closest to the setting you specified.
  - ServerWaitTimeOut
  - ServerWaitPeriods
  - ServerAnimateTimeOut
  - ServerSyncTimeOut
  - ChildRatioThreshold

Preference Settings

This section describes the preferences you can set for rsserver. The settings are grouped into the following categories. Each preference category is described after the table:

Preferences Category	Description
Directory	Controls the locations where rsserver reads data and writes results.
File	Controls how many open files are permitted and whether variable names can be used when specifying file names.
Log File	Controls where and how rsserver writes information to the log file

Preferences Category	Description
Warning	Controls whether rsserver issues warnings about potential incremental update problems and ratios between categories and their descendants.
Output	Controls how often rsserver creates checkpoints for recovery from severe errors during cube creation.
Available Memory	Controls how much memory is available to rsserver when creating cubes.
Data Source Attributes	Describes the physical attributes of data source files, such as the character used for a decimal point.
Communication	Controls properties relating to communications between Transformer on Windows and Transformer on UNIX.
Date Format	Controls the format in which the date is displayed.

Note: In the following lists of preferences, <path> refers to the directory and file name.

Directory

ModelWorkDirectory=<path>

Specifies where rsserver can create a temporary file while you work on your model. The temporary file can be used to recover a suspended model at strategic checkpoints, should a severe error occur during cube creation. This file has the extension .qy?. (The ? is replaced by the character that is used in your version of Transformer.)

The default path is the value of ModelSaveDirectory.

DataWorkDirectory=<path1;path2;...>

Specifies where Transformer creates temporary work files while generating cubes. It will create multiple files automatically. The location of those files is determined by the list of paths that you specify. The files are created in the order specified in the list of paths.

The default path is the value of CubeSaveDirectory.

Notes

- Distributing files across multiple disk drives can improve performance by reducing disk contention.
- The environment variables TMPDIR, TEMP, and TMP can also determine where Transformer creates temporary files. Transformer uses the first environment variable that is defined.

DataSourceDirectory=<path>

For data source files other than .iqd files, specifies where rsserver searches for the files.

The default path is the current working directory.

CubeSaveDirectory=<path>

Specifies where rsserver saves cubes.

The default path is ModelSaveDirectory.

ModelSaveDirectory=<path>

Specifies where rsserver saves models.

If you are running the rsserver command with an .mdl file and you have specified the -s option, a .py? file is created in this directory. (The ? in the extension .py? is replaced by the character that is used in your version of Transformer.)

The default path is the current working directory.

File

FilenameVariables=FALSE

Determines whether rsserver parses environment variables and how it parses them.

If FilenameVariables is set to TRUE, rsserver looks for and removes names in the file name and directory name strings that are marked with a dollar sign ($), for example, $VARIABLE_NAME or ${VARIABLE_NAME}. If such a name is defined in the environment, rsserver replaces it with the value from the environment.

If the value is the default FALSE, rsserver doesn't parse the variables.

The following rules are observed for environment variable entries:

- Each environment variable must be preceded by a dollar character ($). Optional braces ({}) may enclose the environment variable name.
- The environment variable must be alphanumeric (ASCII) and may contain an underscore (_).
- The variable must already be defined at the time the string is used.
- The special characters $, {, or } may appear in a file name or directory string if they are preceded by the escape character (\). The backslash is removed by Transformer before the string is used, for example, a pair of backslash characters (\\) is replaced by one backslash.
- Variable substitution is not performed on the values of environment variables.

Example 1

If .trnsfrmr.rc contains the following lines, rsserver looks for data source files in the subdirectory called data under your home directory.

```
FilenameVariables=true
DataSourceDirectory=$HOME/data
```

Example 2

MONTH is a variable in a predefined data source file. This variable is to be included in the model and uploaded to the server for processing.

In the Server Data File box on the Data Source property sheet, type the following:

```
${MONTH}data.asc
```

On the host system, define the environment variables FilenameVariables and MONTH:

```
MONTH=March ; export MONTH
FilenameVariables=true ; export FilenameVariables
```

The rsserver program uses the data file called Marchdata.asc when it generates categories or creates cubes.

WorkFileMaxSize

The default value is:

- 1500000000 (1.5GB) for Solaris
- 2000000000 (2GB) for AIX, HP and Tru64

Sets the threshold number of bytes at which Transformer splits its workfiles.

The minimum is 100000000 and the maximum is 2400000000.

MultiFileCubeThreshold

Sets a value that determines one of the following options:

- the creation of a single-file cube that is larger than the imposed single-file limit of 2 GB
- the threshold at which multiple files are created for a very large cube, for example, 30,000,000 if your cube is still less than 2 GB or 1,000,000 to create smaller cubes

This multiple file cube has one file with the .mdc extension and one or more files with the .mdp (multi-dimension partition) extension. The cube can't be compressed.

Default: 0 (multiple file generation disabled)

Log File

LogFileDirectory=<path>

Specifies where rsserver creates the log file.

The default is the current working directory.

LogFileName=<path>

Specifies a file name if you want messages written to a file, rather than displayed on the screen. The file name can include the full path.

The default is the standard output stream.

LogFileAppend=FALSE

Specifies that the rsserver log file overwritten for each new model or cube. A value of TRUE appends the new log data to the existing log file.

LogDetailLevel=4

Specifies the types of messages that are written to the log file. Choose from 0 through 4:

- 0 suppresses logging
- 1 includes severe errors only
- 2 includes severe errors and errors
- 3 includes severe errors, errors, and warnings
- 4 includes severe errors, errors, warnings, and informational messages (default)

You can use the log file to check the status of cube creation. The progress of a cube update is indicated by statements in the file, each containing the following fields:

- date and time (24-hour clock) at which the message was issued
- the message severity
- the message text

The text of each message includes information about processing and timing. Messages marked Timing are especially useful to analyze as a series of processing events. You can do this by importing the log file into a spreadsheet application as a tab and comma delimited file. Since messages containing timing information are formatted with a comma (,), this will produce another column in the spreadsheet. You can then filter on this column to analyze just the Timing messages. For example, in Excel, select the third column and use the AutoFilter command.

You can also use the -r option of the rsserver command to control the types of messages generated.

For information about this option, see "-r log_level" (p. 37).

LoggingFrequency=

Specifies, in minutes, how often messages are written to the log file. There is no default. If you do not specify a value, messages are written to the log file whenever they are generated.

Warning

IncUpdateWarnings=TRUE

Issues warnings when an event is going to take place that will make an incrementally updated cube invalid, for example, deleting a category.

The default value of TRUE means that warnings are issued; FALSE disables warnings.

ChildRatioThreshold=35

Issues a warning if the number of child categories for any parent category exceeds the specified value.

Valid values are 1 through 16384. The default is 35.

Output

MaxTransactionNum=500000

Sets how often a checkpoint is written during cube creation. This is defined as the number of records written to a cube before a new checkpoint is created.

If your data sources are constructed from a database, this value shouldn't exceed the size of your database rollback journal.

The default is 500000.

Available Memory

PPDS_READ_MEMORY=16000

Sets the amount of memory, in kilobytes, that is allocated when loading server cubes.

The default value is 16000. A value of 0 means use the default. The minimum value is 100. If you specify a value of less than 100, the minimum value is used.

PPDS_WRITE_MEMORY=32000

Sets the amount of memory, in kilobytes, that is allocated when writing or updating server cubes.

The default value is 32000. A value of 0 means use the default. The minimum value is 100. If you specify a value of less than 100, the minimum value is used.

PPDS_FLUSH=500

Sets the percentage of memory that is released whenever the cache is full. The least recently used records are selected. The value is specified in a .01% scale, for example, 500 is interpreted as 5%.

The default value is 500. A value of 5000 or more (>50%) is automatically changed to 50%. A value of 99 or less (<1%) is changed to 1%.

Note: Memory preferences can only be set as environment variables.

Data Source Attributes

DecimalPoint=.

Specifies the character used as a decimal point in a data source. The default is a period (.).

DefaultSeparator=,

Specifies the field delimiter in a delimited-text data file. The default is a comma (,).

ThousandSeparator=,

Specifies the character used as a thousands separator in a data source. The default is a comma (,).

CenturyBreak=20

Specifies the cut-off date that determines whether the two-digit year (YY) in a six-digit date is a 20th or 21st century date. Transformer interprets values *below* the cut-off as 21st century dates and values *at or above* the cut-off as 20th century dates.

The default is 20. Transformer treats 00 to 19 as 21st century dates and 20 to 99 as 20th century dates. You only need to change the default if your data source includes dates from 1900 to 1919. For example, setting `CenturyBreak=18` means that the values 00 to 17 are interpreted as 2000 to 2017 and the values 18 to 99 are interpreted as 1918 to 1999.

Communication

PowerGridBlockSize=16384

Specifies the size of block, in bytes, used to send information back and forth between Transformer on Windows and Transformer on UNIX.

The minimum block size is 1000. If you specify a smaller block size, 1000 is used.

The maximum block size is 32000. If you specify a larger block size, 32000 is used.

A larger block size speeds the transfer of newly-generated categories from a server model to a client model. If too large a block size is used, a message indicating memory allocation failure appears in the Transformer on Windows window during client-server communications. To resolve this problem, make more virtual memory available in the client or reduce the block size on the server.

ServerWaitTimeOut=10

When multiplied by ServerWaitPeriods, determines the maximum length of time rsserver remains idle before shutting down.

If the network is very busy, or if you set long delays between commands, you may want to adjust ServerWaitTimeOut and ServerWaitPeriods to reduce time-outs. A value of -1 causes rsserver to wait indefinitely.

Valid values are -1 through 32767 seconds. The default is 10 seconds.

ServerWaitPeriods=30

See the previous discussion for ServerWaitTimeOut.

Valid values are 0 through 32767 seconds. The default is 30 seconds.

ServerAnimateTimeOut=3

Determines how long rsserver waits for a response to messages that it sends to Transformer on Windows. These messages appear in the status window that Transformer on Windows displays to indicate that rsserver is working on a command. When the time expires, rsserver disconnects from the client.

If the network is very busy, you may want to increase this value to reduce the number of disconnects; however, doing so may adversely affect throughput if client and server are usually slow to respond.

Valid values are 1 through 32767 seconds. The default is 3 seconds.

Tip: You don’t need to stay attached to the server to monitor its activity. Use the Server Status command in the Server menu in Transformer on Windows to reconnect and reopen the status window as needed.

ServerSyncTimeOut=10

Determines how long rsserver waits for a response from Transformer after sending model information. When the time expires, rsserver shuts down. A value of -1 causes rsserver to wait indefinitely.

Valid values are -1 through 32767 seconds. The default is 10 seconds.

Date Format

LunarFiscalLabeling=TRUE

Determines whether users will be able to view dates in a cube in PowerPlay in lunar year format.

The default value of TRUE indicates that the dates will be displayed in this format. A value of FALSE indicates that dates will be displayed in calendar year format.

Environment Variables

Three types of environment variables affect this product:

- PowerPlay Transformer UNIX Edition
- RDBMS
- shared library

These variables are listed in the following tables.

PowerPlay Transformer UNIX Edition Environment Variables

PYA_LIBRARY, PYA_LOCATION and PYA_USER variables that were present in previous releases are no longer required to run Transformer. However, Transformer shell scripts still define these variables to ensure backward compatibility with client scripts.

COGNOS_HOME

The critical environment variable used by PowerPlay Transformer UNIX Edition.

Sets the directory where Cognos binary files are installed.

Default: cern/bin

***Note:** *n* is the Cognos rendition number (one digit). For example: cer3.

RDBMS Environment Variables

If your application uses a relational database, you must set the RDBMS environment values before sourcing the setpya.sh file.

If you are launching Transformer on UNIX from Transformer on Windows, you must also provide certain database software definitions for Transformer on UNIX.

Note that the values in this table are only examples. Contact your database or network administrator for the correct values for your system.

RDBMS	Environment Variables
Oracle	ORACLE_HOME Defines the top level directory where the Oracle client software (or the entire database install) resides. Example: /mount/app/oracle/product/7.3.3 TNS_ADMIN Defines the directory where the Oracle file tnsnames.ora is found. This file enables calls to the Oracle database to determine which server to connect to. Example: $ORACLE.HOME/network/admin
Sybase	SYBASE Defines the top level directory where the Sybase client software (or entire database install) resides. Example: /sybase DSQUERY Defines the default Sybase server to connect to. Example: /TEST1
Informix	INFORMIXDIR Defines the top level directory where the Informix client software (or entire database install) resides. Example: /usr/informix INFORMIXSERVER One model cannot have data sources from two different Informix servers. Example: coral
DB2	DB2DIR Defines the top level directory where the DB2 client software resides. Example: /usr/db2

Shared Library Environment Variables

To run rsserver, the UNIX loader typically requires the library path environment variable to specify the locations of the Cognos shared libraries, COGNOS_HOME, and, if necessary, the RDBMS shared libraries. The library path environment variable is defined in the following table.

Operating System	Environment Variable
Compaq Tru64 UNIX	LD_LIBRARY_PATH
Sun Solaris	LD_LIBRARY_PATH
IBM AIX	LIBPATH
HP-UX	SHLIB_PATH

The Cognos script setpya.sh, residing in $COGNOS_HOME, sets the library path environment variable to include $COGNOS_HOME as well as any relational database library directories such as $SYBASE/lib, $ORACLE_HOME/lib, etc.

Chapter 5: Run Transformer on UNIX from the Command Line

This chapter describes the rsserver program and its syntax.

The topics covered are

- Run rsserver from the Command Line
- Syntax for rsserver
- Command-line Options List
- Use the Command-line Options
- Examples

Run rsserver from the Command Line

You may run Transformer on UNIX from Transformer on Windows (rsserver) by means of PowerGrid, or you can run rsserver from the command line if you want to set up a production environment on a server to process source data and create cubes.

When you run rsserver from the command line, you use command-line options to

- specify the server-based model that is to be created or updated
- specify processing information

Note: The man page for the rsserver command-line options is in cer*n*/documentation/man/<language>. *n* is the Cognos rendition number (one digit). For example: cer3/documentation/man/english.

When you use rsserver to create cubes you can check the status of the cube by using a shell script including rsserver commands. The exit status of rsserver can be checked to detect operations that did not end successfully. When you run rsserver from the command line and cube creation succeeds, a zero exit status value is returned; or if it fails, a non-zero error code is returned.

The following example illustrates the syntax necessary to determine if rsserver returns an exit status of zero:

Bourne Shell:

```
#!/bin/sh
if rsserver <command-line options>
    then
    #perform action a if exit status is 0
    else
    #perform action b
fi
```

Syntax for rsserver

The syntax for rsserver can be either of the following:

- rsserver options model_file
- rsserver model_file options

where

- rsserver is the executable name.
- model_file is the .py? or .mdl file that rsserver is to process. You must either supply a file name or use the "-" option to have rsserver accept input from the standard input stream.
- options are the command-line entries that provide processing information for rsserver. The options are case sensitive. You must supply at least one option, but you can include more than one.

  For information about command-line options, see the command-line options in the table below.

Command-line Options List

This table lists the command-line options. See the sections following the table for details:

Meaning	Option
generate categories and create cubes	-c
override user preference setting	-D preference_var=setting
update model, but not data	-e
set user-defined preference file	-F preference_file
open specified .py? model and restart failed process from beginning	-i py_model_file
specify database signon	-k signon=userid/password
open specified .mdl file or accept Model Definition Language (MDL) statements	-m mdl_file
open specified .py? file	-p py_model_file
specify log level detail	-r log_level
save model	-s
set current period	-t category_code
get partition status	-u cube_name
specify number of records for test cube	-v data_subset_number
expect input from the standard input stream	- (dash)

For information about the Model Definition Language, see the *Transformer MDL Reference.*

Use the Command-line Options

-c

Generates categories and creates cubes after rsserver loads a model file, interprets MDL statements, or both.

You must use this option with the -p, -m, or -i option.

-D preference_var=setting

Overrides a user preference setting. If you specify the -D option after the -F option, the -D setting overrides the -F file, and vice versa.

For information about user preferences, see "Control Transformer on UNIX with Preferences and Environment Variables" (p. 23).

-e

Updates all the cube metadata that is defined in the model, but does not update the data. The metadata consists of object names, labels, short names, descriptions, drill-through reports, and user classes.

You cannot use this option with -c.

-F preference_file

Specifies the user-defined preference file that rsserver is to use. The name may include a directory path. If you specify the -F option after the -D option, the -F file overrides the -D setting and vice versa.

-i py_model_file

Restarts a failed process when used with another option, such as -c. Unlike the -p option, -i ignores the .qy? checkpoint file and restarts processing from the beginning. The model file is saved after processing.

Use this option with a binary model file (.py?). You cannot use it with the -s option.

-k signon=userid/password

Specifies one or more signons, each consisting of a user ID and password that are needed to access databases.

The signon name cannot contain the ASCII character =.

The user ID cannot contain the ASCII character /.

Signons are required for source data accessed via Impromptu Query Definition (.iqd) data sources, unless they are embedded in the model.

When you insert an .iqd data source into your model, Transformer automatically creates a signon and associates it with the data source. The signon name that Transformer assigns is created from the logical database name in the Cognos.ini file. This logical database name matches the logical database name defined in Impromptu. You can view these signons in Transformer on Windows, but you can’t modify them. Multiple .iqd data sources can use the same signon object.

When you define a signon in a model, the user ID and password are embedded in the model. Passwords are encrypted and saved to .py? files, but they are only saved to .mdl files if Transformer generates verb MDL. When you process an .mdl file, the -k option enables you to specify the user IDs and passwords that you require to access databases. These signons must match the signons that you defined in your Transformer model.

Example

Transformer reads your data source for the server sales model Xyzsales.mdl from an Oracle database via an .iqd file. You have created a signon called sal_log. This signon includes the Oracle user ID corpadm and the password my_pass. To process the .mdl file for sales model Xyzsales.mdl, you enter the following command:

```
rsserver -c -s -m Xyzsales.mdl -ksal_log=corpadm/my_pass
Secure User IDs and Passwords
```

If you use the -k option to pass user IDs and passwords to rsserver, there is a possibility that security breaches may occur. For example, your security can be violated if other users can view the details of your rsserver process via the UNIX ps command or by reading the log file.

To prevent security problems, you have three options:

- store the -k option with the user ID and password information in a file in a secure directory and call the file from the rsserver command line
- embed the user ID (or the user ID and password) in the MDL model file
- create an MDL script file

For the first option, you create a file called, for example, Sal_id.txt, and store it in a secure directory. The file contains the -k option with the user IDs corpadm and corpis and the passwords my_pass and bld_cube.

```
-ksal_log=corpadm/my_pass -ksal_cube=corpis/bld_cube
```

You call the Sal_id.txt file from rsserver as follows:

```
rsserver -c -s -m Xyzsales.mdl `cat Sal_id.txt`
```

For the second option, you add MDL statements to the end of the .mdl model file. The statements update the signon information needed to log on to the database. To embed both the user ID and password, enter the following statement:

```
SignonUpdate "sal_cube" PromptForPassword False UserID "corpis" Password "bld_cube"
```

To embed the user ID only, enter:

```
SignonUpdate "sal_cube" PromptForPassword True UserID "corpis"
```

You can then run rsserver with the -m option, specifying the modified .mdl file without the -k option.

For the third option, you create a secured temporary MDL script on the server. The MDL script updates the model signon, as shown in this example:

```
OpenPY "Xyzsales.py?"
SignonUpdate "sal_cube" PromptForPassword False Password "bld_cube"
SavePY "Xyzsales.py?"
```

To use this MDL script, run rsserver with the -m option, specifying the name of the .mdl file.

-m mdl_file

Specifies an ASCII model or script file (.mdl) to interpret.

If you use multiple occurrences of -m, .mdl files are processed in the order of their occurrence. In addition, you can use - as an argument to accept MDL verb statements from the standard input stream.

To save a file in .mdl format, use -m and the MDL verb SaveMDL, as shown in the following examples:

- Create a separate file, Savemdl.mdl, containing the line
  ```
  SaveMDL "Xyznew.mdl"
  ```
  Then, using the ASCII model file Xyzsales.mdl, enter
  ```
  rsserver -m Xyzsales.mdl -m Savemdl.mdl
  ```
- Create a separate script file Savemdl.mdl, containing the line
  ```
  SaveMDL "Xyznew.mdl"
  ```
  Then, using the binary model file, Xyzsales.py?, enter
  ```
  rsserver -p Xyzsales.py? -m Savemdl.mdl
  ```
- Input SaveMDL from the standard input stream. To do this, first enter the following line on the command line:
  ```
  rsserver -p Xyzsales.py? -m-
  ```
  Then, from the keyboard, enter
  ```
  SaveMDL "Savemdl.mdl"
  ```
  Once you have completed your entries to rsserver, enter the UNIX end-of-file command (Ctrl-D) and rsserver will save the .py? file in .mdl format.

-p py_model_file

Processes the model.

The rsserver program loads the binary model file (.py?) and starts processing either from the last checkpoint saved in the checkpoint file (.qy?), if this file exists, or from the beginning of the .py? file. The program saves changes at termination.

Do not use the -p option with the -s option.

For information about how to restart a failed job from the beginning, see "-i py_model_file" (p. 35). For information about how to save a model, see "-s" (p. 37).

-r log_level

Sets the types of messages that are written to the log file, where log_level is a digit from 0 through 4. Each level includes the errors and messages for all higher levels.

- 0 suppresses logging
- 1 includes only severe errors and above
- 2 includes error messages and above
- 3 includes warning messages and above
- 4 includes informational messages and above (default)

-s

Saves changes to your model upon exit. Changes are saved in a binary model file (.py?).

Do not use this option with -i or -p.

For information about -i, see "-i py_model_file" (p. 35). For information about -p, see "-p py_model_file" (p. 36).

-t category_code

Sets the current period in all manual time dimensions to the date associated with the category. Category_code is the category code of a category in a time dimension.

This option applies only to dimensions that are not set to Automatically Set Current Period in the Time tab (Dimension property sheet) in Transformer on Windows.

Enclose the argument in quotation marks if it contains a tab or space character.

-u cube_name

Writes cube partition information to the log file.

A cube must exist before its partition status can be reported.

To obtain partition information for cubes in a cube group, you must specify the name of an individual cube, not the cube group name.

-v data_subset_number

Specifies how many data source records rsserver should use to create a test cube. If you have a large data source file, this option enables you to do a test run on a limited number of records before processing the entire file.

If the number of records you specify is greater than the total number of records in the file, rsserver treats the process as a normal run, not a test, and uses the whole file.

Use this option with the -m or - p option.

- (dash)

Use this option alone when you want rsserver to accept input from the standard input stream. All options to the right of this option are ignored.

Examples

Save Changes to a Model File

This command starts rsserver, parses an ASCII model file (.mdl), and saves the changes in a binary model file (.py?):

```
rsserver -m go_sales.mdl -s go_sales.py?
```

Generate Categories and Create Cubes

In both examples, the commands start rsserver and build the cubes that the model specifies. The .mdl file is a full model definition.

- This command processes a binary model file (.py?) called go_sales.py?:
  ```
  rsserver -c -p go_sales.py?
  ```
- This command processes an MDL text file called go-sales.mdl:
  ```
  rsserver -c -m go_sales.mdl
  ```

Choose a Preference File

This command tells Transformer to parse an .mdl file using the preference file mypref.prf:

```
rsserver -F mypref.prf -m go_sales.mdl
```

Override Preference File Settings

This command overrides the default value at which rsserver issues warnings that a category has too many descendants:

```
rsserver -D ChildRatioThreshold=25
```

Create a Test Cube from a Subset of Records

This command processes 525 records from binary model file Xyzsales.py?, generates categories, and creates a cube.

```
rsserver -c -p Xyzsales.py? -v 525
```

Combine Options

This command starts rsserver and

- opens the binary model file go_sales_jan.py?
- processes the commands in the .mdl file monthly_update.mdl
- obtains preferences from a file named trnsfrm_prd.prf
- saves the model

```
rsserver -p go_sales_jan.py? -m monthly_update.mdl -F trnsfrm_prd.prf
```

Chapter 6: Production and Maintenance

This chapter describes production and maintenance on the server.

The topics covered are

- Manage Production in a Client-Server Environment
- Client-Server Production Issues
- Schedule Server Production
- Incremental Updates
- Use Transformer on Windows to Check rsserver Status
- Check Job Completion
- Use the Log File
- Check Cube Status
- Restart a Failed Process from a Checkpoint
- Restart a Failed Process from the Beginning

Manage Production in a Client-Server Environment

Note: For information about PowerPlay production in general, see the PowerPlay *Administrator's Guide*.

There are several ways to manage production in a client-server environment:

- Make changes in Transformer on Windows and update your models and cubes on the server. Since client and server models are synchronized whenever you connect to a server model from Transformer on Windows, the models remain identical.

  For information about synchronization, see "Synchronize Models" (p. 19).
- Shift production to Transformer on UNIX. Use a scheduling utility such as cron, to process new data and build cubes on a regular basis. You can also set up MDL scripts to define events that rsserver must perform regularly.
- Use Transformer on Windows to develop local models and then save them in .mdl format. Use ftp (or another file transfer mechanism) to copy the files to the server, where you can use rsserver to process them and create cubes.

  Note: You can transfer only local models that have been saved in .mdl format. Model files saved in .py? format are not binary compatible with model files on the server. (The ? in the extensions .py? and .qy? are replaced by the character used in your version of Transformer.)

If you use rsserver to create cubes for your PowerPlay users, your production environment might include automatically mailing the cubes to their intended audiences. For example, you can set up a cron job that creates a cube and sends the cube to its intended users.

Client-Server Production Issues

If you create large cubes or large numbers of cubes, you are likely to shift routine cube production from Transformer on Windows to Transformer on UNIX. However, if you are updating a cube, the easiest way to do it is to use a client model in Transformer on Windows, rather than a server model in rsserver.

For example, it is easy to add a manual level to a dimension if you use Transformer on Windows. To do the same thing on the server, you need to use MDL to create the manual level, its categories, and the categories you want to connect to the categories in the manual level. You need to know MDL syntax as well as the objects in the model (such as the drill categories and the parents of the categories or levels you are creating).

Schedule Server Production

If you use UNIX scheduling commands, the UNIX at command, or a crontab file, you can schedule batch jobs that run rsserver to create cubes.

For example, you have a model that is used to create 20 individual cubes and a cube group consisting of 10 cubes. You can supply scheduling information to enable and disable the creation of specific cubes at specific times.

Incremental Updates

When you update cubes incrementally, there are special circumstances that you must consider.

If you update a cube incrementally on UNIX, which results in changes to the server model, you must synchronize the server model with the associated client model.

For information about synchronizing models, see the Transformer online help.

If a cube update fails, there are cases when you should not attempt to repeat the most recent incremental update.

For example, when processing fails, some records from the increment may already have been written to the cube. If you restart the process, rsserver adds these records to the cube twice, which results in inaccuracies in the cube.

For complete information about updating a cube incrementally, see Chapter 5 in the *PowerPlay Administrator's Guide.*

Sample MDL Model Update

This example demonstrates how to use MDL scripts to split the generation of categories and the creation of cubes. The following MDL file contains the verb statements that are required to generate categories for the model go_sales.py?, without creating the cubes for that model.

```
OpenPY "go_sales.py?"
PopulateModel
SavePY "go_sales.py?"
```

If these statements are saved in a file called Gen_nat.mdl, you can process them by running rsserver as follows:

```
rsserver -mGen_nat.mdl
```

Use Transformer on Windows to Check rsserver Status

If you start rsserver from Transformer on Windows, you can monitor the server activity from Transformer on Windows. Use the Server Status command in the Server menu. The information is updated every few seconds.

Check Job Completion

When you use rsserver to create cubes, either from Transformer on Windows or the command line, you can check the status of the cube:

- On the server, view the contents of the log file, which contains messages issued by rsserver.
- If you used a crontab file or scheduled the job with the UNIX at command, check your email for a completion message. You can then review the log file for any warnings or errors.
- If you submitted a cube creation job on the server from Transformer on Windows, click the Server Status command in the Server menu to check the current status of the cube creation.
- In Transformer on Windows, click cube Status from the Tools menu.
- Check the ModelSaveDirectory for a checkpoint file (.qy?). Because rsserver automatically deletes checkpoint files when processing ends successfully, a checkpoint file means that a suspended model exists. Check the log file for errors associated with the processing of that model.
- If you created a shell script that includes rsserver commands, you can check the exit status of rsserver to detect operations that did not end successfully. An exit status value of 0 indicates successful completion. Any other value indicates an error.

Use the Log File

By default, all messages generated by rsserver are directed to the standard output stream. You can direct them to a log file instead and control the properties of that log file with preferences that you set in your trnsfrmr.rc or .trnsfrmr.rc files.

For information about the log file, see "Log File" (p. 27).

You can use log file error and warning messages to help you isolate problems encountered as rsserver read the source data or wrote information to the cube.

Check Cube Status

When you use any of the following Server menu commands, Transformer on Windows sends requests for processing to rsserver:

- Generate Server Categories
- Generate Categories from Selected Server Query
- Create Server PowerCubes command
- Create Selected Server PowerCube
- Update Server PowerCubes
- Update Selected Server PowerCubes

When the request is completed, you can review the current status of cubes that have been created or updated on the server.

Steps to Review Current Status of Cubes

1. In Transformer on Windows, open the client model associated with the server model that was used to create the cube.
2. From the Server menu, click Synchronize to connect to the server. Type your server password when prompted.
3. From the Tools menu, click PowerCube Status.

   If a cube is marked Invalid, you may want to reset the status to Warning or OK and make the cube available for access anyway. If the cube is set to Invalid because of an error, you must recreate it.

   For incremental updates, you must identify and correct the problem, restore the cube and its associated model from backup, and then restart the update job.
4. If exceptions are indicated, read the log file.

For information about cube status and actions to take in response to a specific status, see Chapter 5 in the *PowerPlay Administrator's Guide*.

Restart a Failed Process from a Checkpoint

As rsserver processes data on the server, it maintains a checkpoint file with the extension .qy?. When a cube create or update fails, rsserver can use the .qy? file to restart processing at the point of failure.

For example, you are running a quarterly model update with new data and rsserver is unable to locate one of the data source files for one of the data sources in the model. As a result, the model update fails. Use the checkpoint file to restart processing at the point of failure.

The following command tells Transformer on UNIX to restart .py? model file processing.

```
rsserver -p go_sales.py?
```

If an associated .qy? file is found, it is used to restart the process.

Checkpoint and restart are most significant for cube and cube group creation.

For information about checkpoint files, see Chapter 5 in the *PowerPlay Administrator's Guide*.

Restart a Failed Process from the Beginning

When rsserver builds cubes, some conditions may prevent processing from ending successfully. For example, if rsserver has insufficient disk space for the model files, data temporary files and cubes, an error occurs and processing stops. In such a case, use rsserver to start the process from the beginning.

The following command tells Transformer on UNIX to load the saved model file go_sales.py?, but to ignore the last interrupted cube creation process and any associated .qy? files:

```
rsserver -i go_sales.py?
```

For more information about checkpoint files, see Chapter 5 in the *PowerPlay Administrator's Guide*.

Note: If an incremental update fails, follow special steps to ensure that no data is added to the cube more than once.

For information about how to handle incremental update failures, see Chapter 5 in the *PowerPlay Administrator's Guide*.

Appendix A: Configuration Checklist for a Client-Server Environment

This appendix lists the steps to follow when you install and set up PowerPlay in a client-server environment.

The topics covered are

- Set Up a Client-Server Environment
- Install PowerPlay
- Prepare Transformer on Windows
- Prepare Transformer on UNIX

Set Up a Client-Server Environment

To set up a client-server environment in PowerPlay

1. Install PowerPlay locally on your computer and on the server
2. Prepare Transformer on Windows, by creating a prototype, testing it, and changing settings from client to server
3. Prepare Transformer on UNIX, by setting preferences and defining Transformer on UNIX and database-specific environment variables

Install PowerPlay

1. Install PowerPlay from the CD or LAN on your personal computer. For more information, see the *Installation Guide for PowerPlay Transformer Edition (UNIX)*.
2. Install PowerPlay and PowerGrid on the server. For more information, see the *Installation Guide for PowerPlay Transformer Edition (UNIX)*.

Prepare Transformer on Windows

Steps to prepare the server for Transformer on UNIX

1. Start Transformer on Windows.
2. Build the model.
3. Set up the data sources.
4. Create the cube.
5. Make sure that the model is working as expected.

   At this point, you must change certain properties before you can upload the model to the server.
6. In the Source tab of the Data Source property sheet, change the Data Source Location to Server. If you are using multiple data sources, change the location for the remaining data sources, as required.
7. In the Processing tab of the PowerCube property sheet, change the Processed location to On the Server.
8. In the Output tab of the PowerCube property sheet, type the cube file name.
9. In the boxes below Database Type, type the remaining database connection information.

10. When the database signon appears in the Signons list, click the signon, and then type your user ID and password.
11. In the Server tab of the Model Properties item on the File menu, type the model path and the connection information.
12. Confirm the PowerGrid network daemon communications port number. The port number on the client should match the port number on the server where you start the network daemon. The default is 1526, but check with your network administrator.
13. From the Maintenance submenu of the Server menu, click Restore Server Model from Client.

For more information about building a model, setting up data sources, and creating cubes, and about property sheets and menus, refer to the Transformer online Help.

Prepare Transformer on UNIX

Steps to Prepare Transformer on UNIX

On the server, before you run Transformer on UNIX:

1. Set the required preferences and environment variables for Transformer on UNIX.
2. Set the database-specific environment variables.

For information about preferences and environment variables, see "Preference Settings" (p. 24), "Environment Variables" (p. 30), and "RDBMS Environment Variables" (p. 30).

Once you have performed the steps in this checklist, your client-server environment is set up and you can run Transformer on UNIX.

Index