COGNOS®

Server Transformer Guide

Cognos PowerPlay®

BETTER DECISIONS EVERY DAY™

While every attempt has been made to ensure that the information in this document is accurate and complete, some typographical errors or technical inaccuracies may exist. Cognos does not accept responsibility for any kind of loss resulting from the use of information contained in this document.

This page shows the publication date. The information contained in this document is subject to change without notice. Any improvements or changes to either the product or the document will be documented in subsequent editions.

PowerPlay version 6.0

This edition published 1998

Table of Contents

Welcome

The PowerPlay *Server Transformer Guide* contains the information you need to use the Administrator Server 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 PowerCubes
- information about managing the server Transformer environment with user preference files and environment variables
- details about how to run server Transformer from the command line
- guidelines for production and maintenance

This chapter includes

- Who This Book is For
- Other Documentation
- Your Comments are Welcome

Who This Book is For

To use this book effectively you should know

- how to design and build a PowerPlay system
- the UNIX operating system

Other Documentation

A list of the PowerPlay documentation, the *PowerPlay Documentation Roadmap*, is available from the Windows Start menu or the PowerPlay Help menu.

Your Comments are Welcome

We are interested in your comments or questions about the documentation. Please send email to: bipubs@cognos.com.

1

Chapter 1: Overview of Server Transformer

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

The topics covered are

- ✔ Why Use Server Transformer
- ✔ Components of Server Transformer
- ✔ Server Transformer and Standard PowerCubes
- ✔ Server Transformer and Relational Databases
- ✔ Design Server Models and PowerCubes
- ✔ The Client-Server Communications Process

Why Use Server Transformer?

Server Transformer offers additional capabilities to those available with Client Transformer, including

- the ability to handle large data sources. The server can easily process large volumes of data without transferring the data for local processing.
- easier integration of PowerCube builds and data warehouse updates. It is easier to integrate these processes if they both run under the same operating system.
- increased performance. Depending on the size of the server, server Transformer may be able to generate models and cubes faster than local processing on a computer.
- scheduled cube creation. Using scheduling utilities such as cron, you can automate server cube creation so that it is performed during off-peak periods.

- production support. The models and PowerCubes created on the server are subject to automated backup and recovery procedures already in place for the server.
- faster processing. If the server is closer to the data, as in the case of data stored in a database, processing performance improves.

Components of Server Transformer

The Administrator Server Edition of PowerPlay includes components such as communications, model prototyping and synchronization, server menu commands, and configuration, which are available exclusively with this edition. You can also run server Transformer from the command line.

Communications

Client Transformer communicates with server Transformer via PowerGrid, a utility that is included with the Administrator Server Edition. The PowerGrid network daemon (netd) must be installed, configured, and running on the computer where server Transformer listens for requests from Client Transformer. All communications from Client Transformer 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" on page 19.

Model Prototyping and Synchronization

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

For information about model prototyping, see the section "Prototype a Model" on page 24. For information about synchronization, see "Synchronize Models" on page 31.

Server Menu Commands

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

For information about these commands, see "Define a Client-Server Connection" on page 19 and "Change Settings to Build Server Models and Create Server PowerCubes" on page 26.

Server Configuration

Various server preference settings control how server Transformer runs.

For information about the server environment configuration, see Chapter 4 on page 35.

Run Server Transformer from the Command Line

You can run server Transformer from the command line (independent of Client Transformer).

For information about the command line syntax, see Chapter 5 on page 51.

Server Transformer and Standard PowerCubes

You can use server Transformer to create standard PowerCubes that are binary compatible with the ones you create in the Windows environment. Once they are created, you can download 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.

Server Transformer and Relational Databases

You can use relational databases on the server to store PowerCubes.

Before you can store PowerCubes in a relational database, you must create the database tables required to hold the PowerCubes. Both the Administrator Database and Administrator Server editions of PowerPlay include Data Definition Language scripts that enable you to create the database structures Transformer requires to store PowerCubes.

For information about scripts, see the Transformer online Help.

The databases in which you can store PowerCubes include the following:

- Oracle 7.3, 8.0 (accessed via SQL*Net)
- Sybase SQL Server 11 (accessed via CT-Lib)
- Sybase Adaptive Server 11.5
- Informix (Online) Dynamic Server 6.0, 6.5
- DB2 Common Server 2.1
- DB2 Universal Database 5.0
- Microsoft SQL Server 6.0, 6.5

Note: To be able to use a database, its entry in the Cognos.ini file must not be set to 0. Check the following section in the Cognos.ini file and remove the 0 as necessary:

```
[PowerPlay Server List]
Oracle=
Sybase=
MS SQL Server=
Informix=
```

Your list may contain other entries, depending on the version of PowerPlay you are using.

The physical location of the database where the PowerCube is stored is irrelevant to server Transformer. In general, any database that can be accessed through your database connectivity software (for example, Oracle SQL*Net) can serve as a storage device for PowerCubes. For example, if server Transformer is installed on an HP-UX computer on which SQL*Net is installed, server Transformer can create PowerCubes in an Oracle instance on an OpenVMS computer.

Note: This assumes that there is an entry in Tnsnames.ora that enables SQL*Net to access the database on the OpenVMS computer.

The same is true for Impromptu Query Definition (.iqd) data sources accessed by server Transformer—any database that is accessible via your database connectivity software is one that server Transformer can access using .iqd files. For example, if SQL*Net on an IBM AIX computer can access a DB2/400 database (via Oracle's Transparent Gateway) on AS/400, server Transformer can access information from that DB2/400 database.

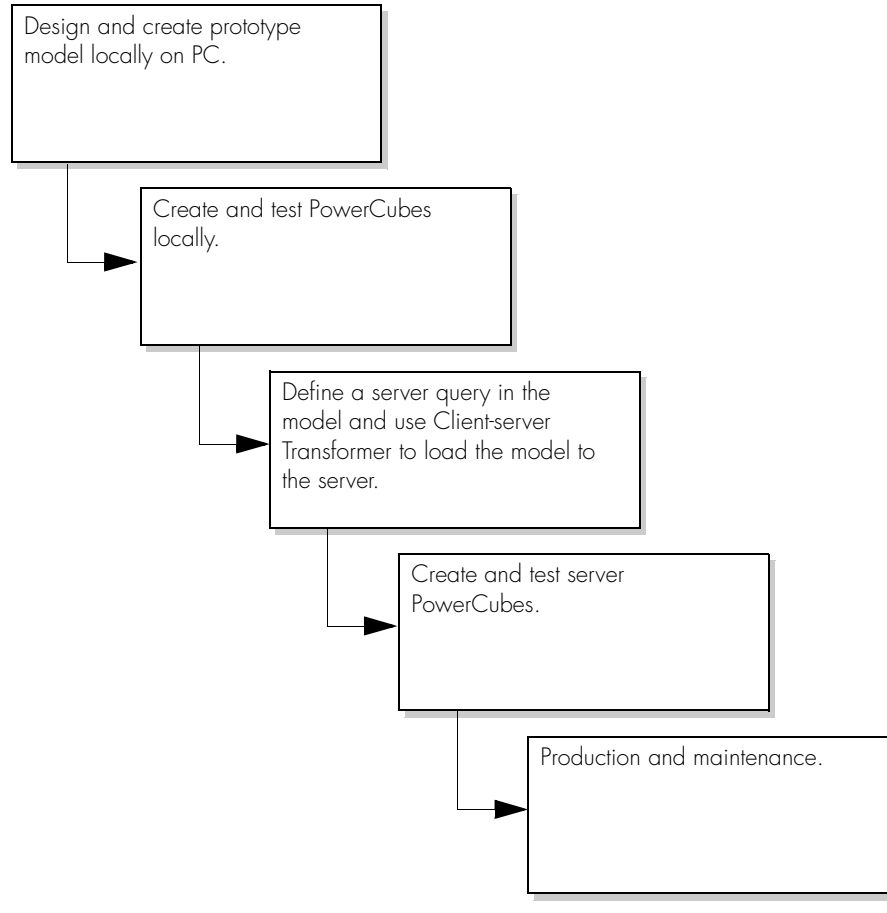
For information about storing PowerCubes in relational databases, see the *PowerPlay Administrator's Guide*.

Design Server Models and PowerCubes

The development of a server model and associated PowerCubes typically begins locally. Using Client Transformer and local queries that are structurally identical to the ones you will use for your server model, you build and test a prototype model and create PowerCubes.

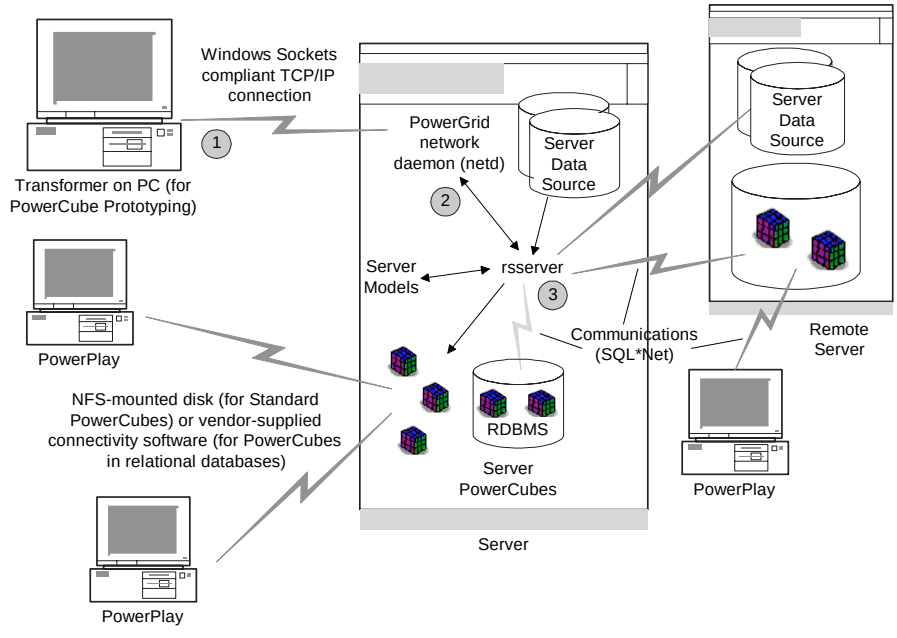
Once you have determined that the local model and PowerCubes are appropriate, you define a server query in the model and store the model on the server. With a server model in place, you can start server Transformer (rsserver) from Client Transformer or from the command line to generate categories for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

Test the server models and PowerCubes 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 PowerCubes and their deployment to users. The following diagram shows the model design and PowerCube creation process.



The Client-Server Communications Process

The following diagram illustrates the client-server communications process for Transformer.



When you use the Server menu to issue commands from Client Transformer to server Transformer, Client and server Transformer communicate according to the following process:

1. Using a Windows Sockets compliant TCP/IP connection, Client Transformer passes the request to the network daemon (netd), the PowerGrid network daemon.
2. The network daemon receives the request, starts server Transformer (rsserver), and connects it to Client Transformer. After that, Client Transformer 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 Client Transformer. 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, server Transformer carries out the requests, such as generating categories for a model or creating PowerCubes, on the server.

An important part of this communication process is synchronization.

For information about synchronization, see "Synchronize Models" on page 31.

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

2

Chapter 2: Set Up Client-Server Transformer Communications

This chapter describes communications between Client Transformer and server Transformer.

The topics covered are

- ✔ PowerGrid
- ✔ Define a Client-Server Connection
- ✔ Server Transformer Shell Scripts

PowerGrid

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

For information about setting up server Transformer and PowerGrid, see the *Administrator Server Installation Guide*.

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

For information about how to issue commands to server Transformer, see Chapter 5 on page 51.

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

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

For information about how to define a connection, see "Define a Client-Server Connection" on page 19.

- the server environment is set up so that server Transformer creates models and PowerCubes where and how you want them created.

For information about the server environment, see "Manage the Server Transformer Environment" on page 35.

For information about the steps to configure a client-server environment, see Appendix A on page 69.

How PowerGrid Works

Following is a description of how PowerGrid sets up communications between Client Transformer and server Transformer:

1. PowerGrid netd is running on the server, listening for messages on the port that was selected at installation time. The default is 1526.
2. From Client Transformer, 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 pya60#\bin.

   Note: # is the build number and can be up to three digits (for example, 113).

   For information about the rsserver.sh script, see "Server Transformer Shell Scripts" on page 21.

3. PowerGrid hears the message and runs rsserver.sh, passing the Client Transformer address to the shell script.
4. The rsserver.sh script starts rsserver, the server Transformer executable.
5. Server Transformer sends a confirmation message to Client Transformer informing it that it has run the script.

For the remainder of the session, Client Transformer communicates directly with server Transformer, 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 Client Transformer
- the Server tab in the Model property sheet (from the File menu, click Model Properties) in Client Transformer
- 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 server Transformer, 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. From the installation directory, 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 Cognos.ini file. By default, Cognos.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 server Transformer 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 server Transformer 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 installed, 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 services file 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.

   ```
   #COGNOS PowerGrid Services
   powergrid 3125/tcp
   ```

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 server Transformer 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 netd, and that netd is responding to them.

Steps to Test a Connection

1. From the installation directory, 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 netd 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 NETD owned by 'root'  [PID='818']
```

If the connection is not working, verify that

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

Server Transformer Shell Scripts

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

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

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

For information about environment variables, see "Environment Variables" on page 46.

3

Chapter 3: Create Server Models and PowerCubes

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

The topics covered are

- ✔ Use Server Transformer on a Remote Computer
- ✔ Prototype a Model
- ✔ Set Up Queries for the Prototype
- ✔ Change Settings to Build Server Models and Create Server PowerCubes
- ✔ Synchronize Models

Use Server Transformer on a Remote Computer

To use Client Transformer to control server Transformer 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" on page 19.

- configure the server environment before you can communicate with server Transformer and create server models and PowerCubes.

For information about configuring the server environment, see Chapter 4 on page 35.

Prototype a Model

Server models often start with a prototype model that you create using Client Transformer. Once the prototype model is developed to your satisfaction, use server Transformer 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 Queries for the Prototype

Models you create using server Transformer must read data sources from the server where server Transformer is running. Because server Transformer 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 queries that provide access to a sufficient subset of the source data to enable you to design your model. You can use any source data type supported by Transformer when creating your prototype model.

The queries 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 server Transformer.
- 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 query than in the server query. If this is the case, data records found in the server query, which were not found in the local queries, will provide new categories for the server model.

Use Supported Data Sources for Server Models

Server Transformer uses the following source data 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.

Notes:

- In Client Transformer, when you are defining a query that will be used by server Transformer, the only valid source data types are those listed above.
- If PowerPlay and Impromptu were installed in different directories and you use .iqd files that support User-Defined Functions (UDF), you must copy the files associated with the UDF to the PowerPlay directory.

For more information about these files, see the Impromptu online book *How to Create User-Defined Functions*.

Use Impromptu to Set Up Queries

The easiest way to set up a local query for your prototype model is to use Impromptu. 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 source data 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 source data contains a lot of duplicate records

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

Steps to Build Prototype and Server Queries

1. In Impromptu, create an .iqd file that returns sufficient data to build a prototype model.
2. In Client Transformer, 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 Client Transformer, 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 Client Transformer, on the Query property sheet, change the Query 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 Client Transformer 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 Client Transformer, you change each query to a server query. You can then generate server categories and create server PowerCubes using the .iqd files with no restrictions set on data retrieval.

Change Settings to Build Server Models and Create Server PowerCubes

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 PowerCube creation are available until you change some model settings. These settings define

- how Client Transformer communicates with server Transformer
- how and where server Transformer builds the server model and creates PowerCubes

For information about the steps involved in configuring a client-server environment, see Appendix A on page 69.

Steps to enable the commands in the Server menu and to send requests from Client Transformer to server Transformer

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

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

If you are using the server to create PowerCubes in a relational database and you have been prototyping locally using standard PowerCubes, you must change the Database Type setting in the PowerCube property sheet for each PowerCube you want the server to create. This enables the server to write PowerCube data to the relational database.

Note: If the Database Type box does not show the database you need, check the following section in the Cognos.ini file. If your database entry is set to 0, remove the 0:

```
[PowerPlay Server List]
Oracle=
Sybase=
MS SQL Server=
Informix=
```

Your list may contain other entries, depending on the version of PowerPlay you are using.

On the server, you can use parameters for server Transformer that control the default location for server PowerCube files.

For information about how to control the server environment, see Chapter 4 on page 35.

Note: When you are prototyping models for PowerCubes that you will store in a relational database, we recommend you first locally create the models and PowerCubes. This way, you can test the cubes without having to set up the relational database environment for them. Later, you can change the PowerCube Database Type property to create your production cubes in a relational database.

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 server Transformer 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 server Transformer.

Provide Server Query Definitions

Server Transformer can process only server queries. By changing the location of your queries from Local to Server, you identify a location where server Transformer can access the server data required to build the server model and create server PowerCubes.

Steps to change the query location from local to server

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

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

Note: If you are generating PowerCubes, the file settings (local or server) for your queries must match the file settings for your PowerCubes (processed locally or on the server). You can use local data sources to generate PowerCubes locally. If you are creating models and PowerCubes using the server, the Queries list must contain at least one server transaction query 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 PowerCube Definition to Server

By providing a server PowerCube definition in Client Transformer, you specify that category generation and PowerCube creation are done by server Transformer.

Steps to change the PowerCube 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 PowerCube file name.
4. Repeat steps 1 to 3 for each cube you want server Transformer 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" on page 31.

Note: To copy your client model to the server using an FTP program, save the model in ASCII (.mdl) format rather than binary (.py?) format. You can't use an FTP program with a .py? file because the Windows and server environments 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 server Transformer processes and stores the model as a server model.

Tip: To have server Transformer create PowerCubes, you can use the Create Server PowerCubes 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 server Transformer 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) 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 queries to generate server categories
- Generate Categories from Selected Server Query, which uses only the selected server query to generate server categories

Both commands cause PowerGrid to start a server Transformer task, or connect to an active server Transformer task, which uses the defined server queries 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 PowerCubes

Once you have created a server model and generated categories for that model using server queries, you can use server Transformer to create the PowerCubes associated with that model.

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

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

Both commands cause PowerGrid to start a server Transformer task or connect to an active server Transformer task, which creates PowerCubes 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 server Transformer for processing. The rsserverda program closes automatically once it has read a query. A separate instance of rsserverda is started for each query that uses multiprocessing. If server Transformer 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 query (usually the QueryMake 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 Client Transformer to keep in step with server Transformer so you can use Client Transformer's 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 Client Transformer interface.

Automatic Synchronization

When you choose a command from the Server menu, Client Transformer 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 Client Transformer, 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 Client Transformer 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 Client Transformer and a server model from the client using the Restore Server Model from Client command. Later, without Client Transformer running, you use the rsserver command to create a server-based PowerCube. If the server data source includes records for which no categories exist in the model, server Transformer creates new categories.

The server model is now updated and the client model is out of date. The next time you start Client Transformer, open the client model, and connect to server Transformer, 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”
```

4

Chapter 4: Manage the Server Transformer Environment

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

The topics covered are

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

Control Server Transformer with Preferences and Environment Variables

When rsserver is called from Client Transformer or from the command line, it populates models and creates PowerCubes, 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 Client Transformer 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 source data files, include an environment variable definition such as the following in the rsserver.sh file:

```
DataSourceDirectory=mydatadir; export mydatadir
```

Example 2

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

```
rsserver -D DataSourceDirectory=mydatadir
```

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 pound 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
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 PowerCube creation.
Available Memory	Controls how much memory is available to rsserver when creating PowerCubes.
Query Attributes	Describes the physical attributes of source data files, such as the character used for a decimal point.
Communication	Controls properties relating to communications between Client Transformer and server Transformer.
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 PowerCube 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 PowerCubes. 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 PowerCubes.

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.

Rules for Environment Variables Entries

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 source data files in the subdirectory called data under your home directory.

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

Example 2

MONTH is a variable in a predefined source data 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 Query 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 progam uses the data file called Marchdata.asc when it generates categories or creates PowerCubes.

OpenFilesLimit=0

Sets the maximum permitted number of concurrently open files.

The default value is 0, which is the system-specified limit of open file descriptors, typically 60 files on HP-UX and 64 files on Sun Solaris.

The maximum is the hard limit value that the system administrator establishes for your system. This is typically 1024 files.

Values greater than 0 and less than 15 become 15.

This preference is ignored on IBM AIX, SINIX, and Digital UNIX.

Note: The system setting MaxUser also imposes a limit on the number of files rsserver can open. You may need to increase this value, since it determines the number of open files allowed for any process.

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 PowerCube creation. The progress of a PowerCube 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" on page 57.

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 queries 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 PowerCubes.

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 PowerCubes.

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.

Query Attributes

DecimalPoint=.

Specifies the character used as a decimal point in a query. 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 query. 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 source data 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 Client Transformer and server Transformer.

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 Client Transformer 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 Client Transformer. These messages appear in the status window that Client Transformer 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 Client Transformer 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 Administrator Server
- RDBMS
- shared library

These variables are listed in the following tables.

PowerPlay Administrator Server Environment Variables

Here are the critical environment variables used by PowerPlay Administrator Server.

Environment Variable	Description
COGNLSTAB	Points to the Cognos NLS settings table. Default: $PYA_USR/etc/coglangtab
COGNOS	Sets the installation directory for Cognos products. This environment variable is used only by the installation program. Default: /usr/cognos
DMDBINI	Points to the location of Cognos database management initialization files. Default: $PYA_USR/etc
PYA_LIBRARY	Identifies the location of shared libraries used at execution time. Default: $PYA_USR/lib
PYA_LOCATION	Identifies the location of the directory containing the executable rsserver and the shell script rsserver.sh. Default: $PYA_USR/bin
PYA_USR	Identifies the installation directory for server Transformer. Default: /usr/cognos/pya60# *
SRVCMSGS	Identifies the message file used by the Cognos services layer and the PowerPlay Data Services message catalog (ppdsmsg.cat). Default: $PYA_USR/msg/english/ srvcmsgs.msg

Environment Variable	Description
LANG	Points to the locale for NLS settings. The locale is set by the system and is a system wide default.
PPDS_SYBASE_PACKET_SIZE	Sets the value for the network packet size that PowerPlay Data Services is to use when opening a connection to a Sybase database. Default: 2048 **Note:** The typical setting for Sybase 11.x is 4096. The minimum setting is 512.

***Note:** # is the build number, and can be up to three digits (for example, 113).

RDBMS Environment Variables

If your application uses a relational database, you must set the RDBMS environment values before sourcing the setpya.csh or setpya.sh files, since the settings of other variables depend on these database variables.

If you are launching server Transformer from Client Transformer, you must also provide certain database software definitions for server Transformer.

For information about these definitions, see "Use Client-Server Transformer with Relational Databases" on page 49.

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 intall) 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 intall) resides.
	Example: /usr/informix
	INFORMIXSERVER
	Defines the default Informix server to connect to. Note that Server Transformer can write an RDBMS cube to an Informix server even if the server has not been defined by this variable.
	One model cannot have queries 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, PYA_LIBRARY, and, if necessary, the RDBMS shared libraries. The library path environment variable is defined in the following table.

Operating System	Environment Variable
Digital UNIX	LD_LIBRARY_PATH
Sun Solaris	LD_LIBRARY_PATH
IBM AIX	LIB_PATH
HP-UX	SHLIB_PATH

Transformer on SNI UNIX doesn't use shared libraries and, therefore, does not require the LD_LIBRARY_PATH environment variable.

The Cognos scripts setpya.sh and setpya.csh, which reside in PYA_LOCATION, set the library path environment variable to include PYA_LIBRARY as well as any relational database library they require such as $SYBASE/lib, $ORACLE_HOME, or both.

Use Client-Server Transformer with Relational Databases

The program rsserver requires definitions for a few UNIX environment variables. This information is needed whether you start rsserver from the shell command line or from Client Transformer via PowerGrid.

The scripts setpya.sh or setpya.csh define most of these. However, if your application uses a relational database, the database software also requires some definitions. If you don't provide them, Transformer won't be able to access the database.

An interactive login normally defines the RDBMS variables in a login script, without user interaction. Since there is no login when you launch rsserver from Client Transformer, you must define the symbols that the Oracle client library or the Sybase or Informix database management libraries need to access the remote database. One way to do this is to define the symbols near the top of script powergrid/tcpip/netbin/rsserver.sh, before the script runs setpya.sh.

The following example shows how this might be done for the Oracle client library. Note that the variables used here are only examples. You must specify the directories for your site. In addition, Sybase and Informix database management libraries require their own environment variables.

```
ORACLE_HOME=/isv/oracle/7.3 ; export ORACLE_HOME
TNS_ADMIN=$ORACLE_HOME/network/admin ; export TNS_ADMIN
LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${ORACLE_HOME}/lib
export LD_LIBRARY_PATH
```

The above example might be suitable for a system where all users access a single database, in this case Oracle, and where they all use the same tnsnames.ora file. To provide more flexibility, you could add the following commands to rsserver.sh just above the point where it runs setpya.sh.

```
#Lets you define environment variables, for example, those
#required for rsserver as an RDBMS client. This permits you to
#customize the environment in the manner of Bourne shell (sh). You
#may comment out these commands if the initialization scripts are
#not required for your site.
if [ -r /etc/profile ] ; then . /etc/profile ; fi
if [ -r ./.profile ] ; then . ./.profile ; fi
```

5

Chapter 5: Run Server Transformer 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

In most cases, Client Transformer starts server Transformer (rsserver) by means of PowerGrid. However, you can also run rsserver from the command line if you want to set up a production environment on a server to process source data and create PowerCubes.

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 pya60xxx/man/english/cat1, where *x* is the PowerPlay build number. The number may be one to three digits.

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.
- 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 "Command-line Options List" on page 52.

- 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.

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
suppress banner	-nologo
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 powercube_name

Meaning	Option
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 Server Transformer with Preferences and Environment Variables" on page 35.

-e

Updates all the PowerCube 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 the following, unless they are embedded in the model:

- source data accessed via Impromptu Query Definition (.iqd) queries
- PowerCubes stored in relational databases

When you insert an .iqd query into your model, Transformer automatically creates a signon and associates it with the query. 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 Client Transformer, but you can't modify them. Multiple .iqd queries 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. Because the PowerCube is stored in an Oracle database, you have also defined a signon called sal_cube with the user ID corpis and the password bld_cube. 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
-ksal_cube=corpis/bld_cube
```

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 -p 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 -m 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.

-nologo

Suppresses the introductory banner.

-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" on page 53. For information about how to save a model, see "-s" on page 57.

-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" on page 53. For information about -p, see "-p py_model_file" on page 56.

-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 Client Transformer.

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

-u powercube_name

Writes PowerCube partition information to the log file.

A PowerCube 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 PowerCube, not the cube group name.

-v data_subset_number

Specifies how many source data records rsserver should use to create a test PowerCube. 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 PowerCubes

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 PowerCube from a Subset of Records

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

```
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
```

6

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 Client Transformer to Check rsserver Status
- ✔ Check Job Completion
- ✔ Use the Log File
- ✔ Check PowerCube 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 Client Transformer and update your models and PowerCubes on the server. Since client and server models are synchronized whenever you connect to a server model from Client Transformer, the models remain identical.

For information about synchronization, see "Synchronize Models" on page 31.

- Shift production to server Transformer. 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 Client Transformer 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 PowerCubes.

  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 standard PowerCubes, your production environment might include automatically mailing the PowerCubes to their intended audiences. For example, you can set up a cron job that creates a PowerCube and sends the PowerCube to its intended users.

Client-Server Production Issues

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

For example, it is easy to add a manual level to a dimension if you use Client Transformer. 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).

Also, you must use Client Transformer for updates if your model has enabled Authenticator user classes.

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 PowerCubes.

For example, you have a model that is used to create 20 individual PowerCubes and a PowerCube 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 PowerCubes incrementally, there are special circumstances that you must consider. Most notably, if a PowerCube 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 PowerCube. If you restart the process, rsserver adds these records to the cube twice, which results in inaccuracies in the PowerCube.

For information about how to avoid problems that result when a PowerCube update fails, see Chapter 5 in the *PowerPlay Administrator's Guide*.

Sample PowerCube Creation Job

This example demonstrates the process required to set up a PowerCube creation job:

- create an MDL script, which contains verb statements that create specific PowerCubes.
- create a C shell script to start rsserver, which runs the MDL script.
- generate crontab entries to run the C shell script at certain times to create the PowerCubes.

The sample model, called go_sales.py?, includes a PowerCube definition for sales in the United States (Usa_sales) and a definition for sales in Europe (Eur_sales).

Sample MDL Scripts

The MDL scripts that are called by the C shell scripts contain verb statements that create specific PowerCubes.

The MDL script called usa_monthly_cube.mdl contains the statements that open the model file, create the cube for USA sales, and save and close the model file:

```
OpenPY “go_sales.py?”
CreateFromCubes OnServer “Usa_sales”
SavePY “go_sales.py?”
```

The MDL script called eur_weekly_cube.mdl contains the statements that open the model file, create the cube for European sales, and save and close the model file:

```
OpenPY “go_sales.py?”
CreateFromCubes OnServer “Eur_sales”
SavePY “go_sales.py?”
```

Sample C Shell Scripts

The C shell scripts start rsserver, which runs the MDL scripts. Use the chmod command to set the shell script files to execute.

The C shell script called create_usa_cubes.csh is used to create the USA sales cube:

```
#!/bin/csh
# set rsserver environment
source $PYA_USR/setpya.csh
setenv LogFileName rsserver_usa_monthly.log
# call rsserver
rsserver -musa_monthlyy_cube.mdl
```

The C shell script called create_eur_cubes.csh is used to create the European sales cube. It is identical to the script used to create the USA sales cube, except that the rsserver command in this case references the MDL file eur_weekly_cube.mdl.

For information about rsserver command-line options, see Chapter 5 on page 51.

Sample Crontab Files

The crontab file contains information about the job you want to run. Each line in the file consists of six fields, separated by spaces or tabs. The first five fields contain scheduling information (minute, 0-59; hour, 0-23; day of the month, 1-31; month of the year, 1-12; day of the week, 0-6, 0 is Sunday). The sixth field specifies the command to run.

To generate the PowerCube for USA sales on the 1 st and 15th of every month and the European PowerCube every Monday, you must create two crontab files to schedule these cube generation cycles.

To run the C shell script create_usa_cubes.csh on the 1 st and 15th of every month, create a crontab file that has the following entries:

```
0 0 1,15 * * .  .profile  ;  create_usa_cubes.csh
```

To run the C shell script, create_eur_cubes.csh file each Monday, create a crontab file that has the following entries:

```
0 0 * * 1 .  .profile  ;  create_eur_cubes.csh
```

Sample MDL Model Update

This example demonstrates how to use MDL scripts to split the generation of categories and the creation of PowerCubes. 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 Client Transformer to Check rsserver Status

If you start rsserver from Client Transformer, you can monitor the server activity from Client Transformer. 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 PowerCubes, either from Client Transformer or the command line, you can check the status of the PowerCube:

- 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 Client Transformer, click the Server Status command in the Server menu to check the current status of the PowerCube creation.
- In Client Transformer, click PowerCube 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" on page 41.

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 PowerCube.

Check PowerCube Status

When you use any of the following Server menu commands, Client Transformer 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 PowerCubes that have been created or updated on the server.

Steps to Review Current Status of PowerCubes

1. In Client Transformer, open the client model associated with the server model that was used to create the PowerCube.
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 PowerCube 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 PowerCube 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 source data files for one of the queries 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 server Transformer 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 PowerCube and PowerCube 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 PowerCubes, some conditions may prevent processing from ending successfully. For example, if rsserver tries to write a PowerCube to a database that is locked, an error occurs and processing stops. In such a case, use rsserver to start the process from the beginning.

The following command tells server Transformer 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*.

A

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 Client Transformer
- ✔ Prepare Server Transformer

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 Client Transformer, by creating a prototype, testing it, and changing settings from client to server
3. Prepare server Transformer, by setting preferences and defining server Transformer and database-specific environment variables

Install PowerPlay

1. Install PowerPlay from the CD or LAN on your personal computer. For more information, see the *PowerPlay Installation Guide*.

2. Install PowerPlay and PowerGrid on the server. For more information, see the *Administrator Server Installation Guide*.

Prepare Client Transformer

Steps to prepare the server for server Transformer

1. Start Client Transformer.
2. Build the model.
3. Set up the queries.
4. Create the PowerCube.
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 Query property sheet, change the Query Location to Server. If you are using multiple queries, change the location for the remaining queries, 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 PowerCube file name.

If you are storing the cube in a database, complete steps 9 to 11; otherwise, skip to step 12.

9. In the Output tab of the PowerCube property sheet, click the database type in the Database Type box.

   Note: If the Database Type box does not show the database you need, check the following section in the Cognos.ini file. If your database entry is set to 0, remove the 0:

   ```
   [PowerPlay Server List]
   Oracle=
   Sybase=
   MS SQL Server=
   Informix=
   ```

   Your list may contain other entries, depending on the version of PowerPlay you are using.
10. In the boxes below Database Type, type the remaining database connection information.

11. When the database signon appears in the Signons list, click the signon, and then type your user ID and password.
12. In the Server tab of the Model Properties item on the File menu, type the model path and the connection information.
13. 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.
14. From the Maintenance submenu of the Server menu, click Restore Server Model from Client.

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

Prepare Server Transformer

Steps to Prepare server Transformer

On the server, before you run server Transformer:

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

For information about preferences and environment variables, see "Preference Settings" on page 38, "Environment Variables" on page 46, and "RDBMS Environment Variables" on page 47.

3. Run the required database script to set up the tables needed for the database cube.

For information about database scripts, see the Transformer online Help.

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

Index

Symbols

—C—

—D—

—E—

—F—

—I—

—J—

—K—

—L—

—M—

—N—

—O—

—P—

—Q—

—R—

—S—

—T—

—U—

—V—

—W—