IBM® DB2® Universal Database

Data Warehouse Center Application Integration Guide

Version 7

SC26-9994-00

IBM® DB2® Universal Database

Data Warehouse Center Application Integration Guide

Version 7

SC26-9994-00

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 303.

Contents

About this book

This book is designed to help developers of data warehousing solutions to integrate their applications with the Data Warehouse Center and the Information Catalog Manager. You can use this book to write programs that transfer and transform an application's metadata into a format that the Data Warehouse Center and the Information Catalog Manager can use. You can also use the information in this book to tailor the format of the Information Catalog Manager.

Who should read this book

This book is intended for developers of data warehousing solutions who are creating an automated interface between another company's data warehousing application and the Data Warehouse Center, the Information Catalog Manager, or both.

You must have some information processing support experience, but might need the assistance of other support personnel in the enterprise at times. You must be familiar with the Data Warehouse Center and the Information Catalog Manager before you use the integration features described in this book. Specifically, you must know how to do the tasks listed in the following table:

Task	**For more information, see:**
Create an information catalog	*Information Catalog Manager Administration Guide*
Import and export metadata	*Information Catalog Manager Administration Guide*
Define a warehouse agent site	*Data Warehouse Center Administration Guide* and the Data Warehouse Center online help
Create, promote, run, and monitor steps	*Data Warehouse Center Administration Guide* and the Data Warehouse Center online help
Create Data Warehouse Center programs and use them in a step	*Data Warehouse Center Administration Guide* and the Data Warehouse Center online help
Modify parameters for Data Warehouse Center programs	*Data Warehouse Center Administration Guide* and the Data Warehouse Center online help

Task	For more information, see:
Import and export metadata	*Data Warehouse Center Administration Guide* and the Data Warehouse Center online help

For a list of publications for the Data Warehouse Center and Information Catalog Manager, see "Bibliography" on page 309.

Part 1. Integrating Applications

Chapter 1. Planning to integrate your applications

You can use the DB2 Universal Database™ Data Warehouse Center to bring together various applications that help users build and manage a data warehouse. You can identify the data that you want to manage and transform that data into information that will be meaningful for warehouse users.

You can use the Data Warehouse Center to provide a variety of information and services to other data warehousing applications, including:

- Providing metadata about source data and target data that is used in the warehouse.
- Transforming data by issuing SQL or by running another warehousing application.
- Scheduling extracts and transformations of data that is based on the date and time or on an event.
- Publishing metadata for warehouse users to use.

When you integrate your applications with the Data Warehouse Center, you provide a single point of control for warehouse administrators, while enabling them to use the best warehousing applications.

How partner applications can work with the Data Warehouse Center and the Information Catalog Manager

In this book, a *partner application* is an application that runs independently from the Data Warehouse Center and provides some kind of support for a data warehousing solution. You can define the application to the Data Warehouse Center to include it in a warehouse-building process that can include multiple applications.

For example, assume that you want to unload operational data from an IMS™ database, clean the data, and load the cleansed data into a DB2® warehouse database. Users then query the cleansed data. You have three partner applications:

- Partner application 1 unloads data from a database, performs simple transformations, such as joining tables, and writes the transformed data to a warehouse database.
- Partner application 2 cleans the data to prepare the data for the warehouse.

- Partner application 3 queries and reports on the data in the warehouse. It contains metadata about the tables in the warehouse that users can search for specific attributes. Users use the metadata to determine which tables have the data that they need.

You use these three applications together in the following process:

1. Partner application 1 extracts data from multiple segments in a source IMS database.
2. Partner application 1 joins the data from the source segments, and writes the joined data to file 1.
3. Partner application 1 writes the joined data to file 1.
4. Partner application 2 reads the data from file 1.
5. Partner application 2 cleans the data by matching names and by using other data cleansing techniques.
6. Partner application 2 writes the cleansed data to file 2.
7. Partner application 1 reads the data from file 2.
8. Partner application 1 writes the data to a warehouse database.
9. Partner application 3 displays the data in the warehouse or reports about the data in the warehouse when users select tables to query.

Figure 1 on page 5 illustrates how the three partner applications work together.

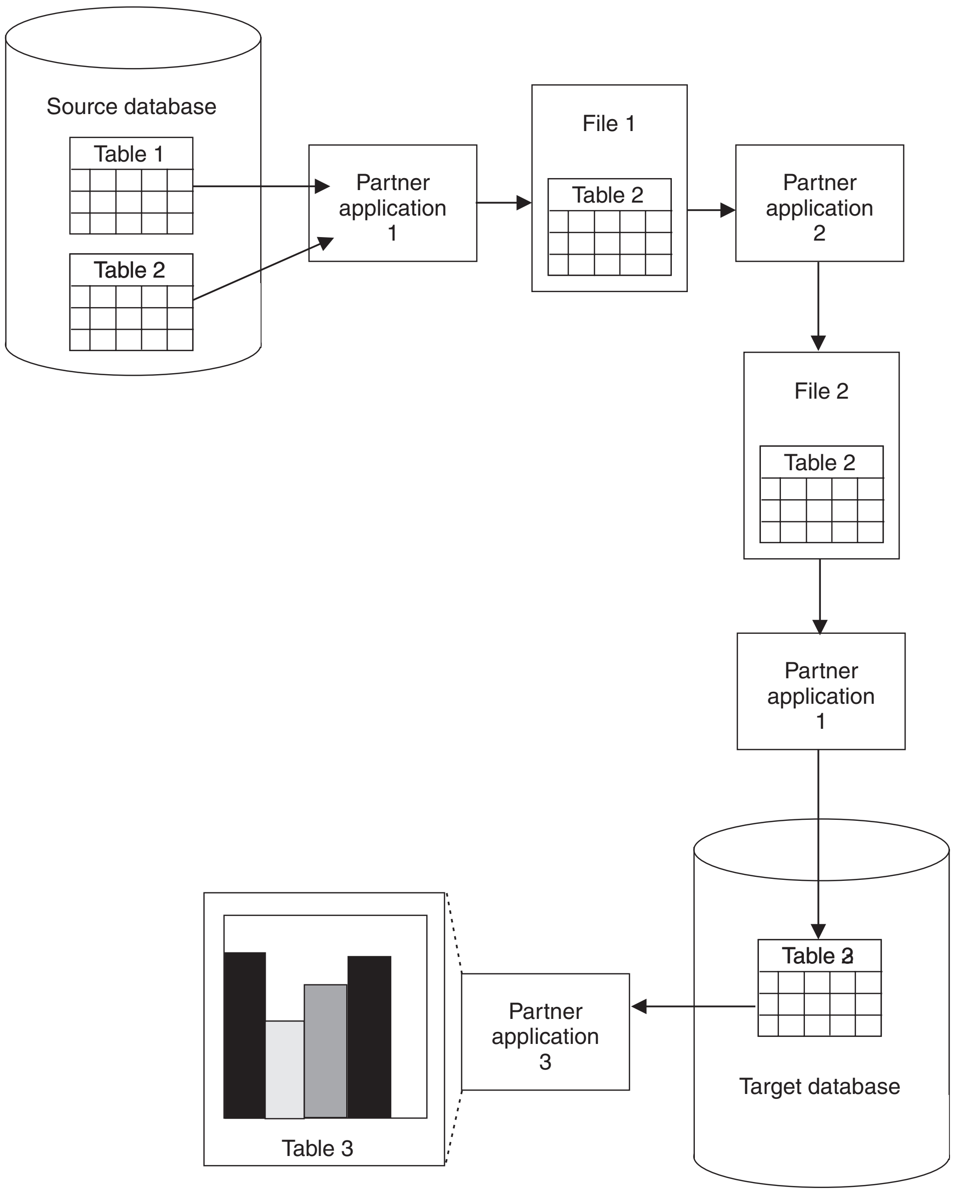


Figure 1. Using partner applications together to build a warehouse

Managing partner applications

You can use Data Warehouse Center steps to manage the process. A *step* is a single operation on data in a warehouse process. In most cases, a step includes a warehouse source, the transformation, or movement of data, and a warehouse target. A step can be run according to a schedule, or it can cascade from another step. You use steps to define and schedule each step in the extraction, transformation, and writing of the data.

A basic step performs the following tasks:
- It extracts data from at least one table or file.
- It uses Data Warehouse Center SQL processing to transform the data, or calls a program that transforms the data.
- It writes the transformed data to a table.

In the partner application example, you define three steps, one for each source-to-target transformation:
- The Unload step performs tasks 1 through 3.
- The Clean step performs tasks 4 through 6.
- The Load step performs tasks 7 through 8.

Because Partner application 3 transforms data at user request in task 9, you do not define a step for task 9.

In the definition of the step, you can schedule a date and time to run the step. At that time, the Data Warehouse Center begins the process that the step defines by issuing SQL statements or starting the program. You can also specify that a second step is to start after the first step finishes processing.

You can schedule the first step to run at a particular date and time. You can schedule the second step to start after the first step runs. You schedule the third step to start after the second step runs. In this manner, you can automate the process of running multiple partner applications.

Managing metadata

To define this process, you import partner metadata into the Data Warehouse Center. In this book, *partner metadata* is metadata that partner applications use and store outside of the Data Warehouse Center.

In the partner application example, you import the following metadata into the Data Warehouse Center:
- From Partner application 1, metadata about the databases, File 1, and the application
- From Partner application 2, metadata about File 2 and the application

You can then export the metadata about the files to the partner applications so that both partner applications use the same information:
- You export metadata about File 2 to Partner application 1.
- You export metadata about File 1 to Partner application 2.

You can also export metadata from the Data Warehouse Center to the Information Catalog Manager to provide information about the data in the warehouse to users of the warehouse. You can import metadata for the sources and targets, as well as the transformations of the data from its source format to its target format. The users of your warehouse can obtain information about the lineage of the data in the warehouse from the metadata that you import.

In the partner application example, you export metadata about the table in the warehouse, Table 3, to the Information Catalog Manager.

You can import metadata into the Information Catalog Manager directly from the Data Warehouse Center. You can also import metadata into the Information Catalog Manager if the partner applications support metadata in MDIS format.

You can export metadata from the Information Catalog Manager to a partner metadata store. In the partner application example, you export metadata about Table 3 from the Information Catalog Manager to the metadata store for Partner application 3. Users view the metadata for Table 3 to determine its contents.

Figure 2 on page 8 shows the flow of metadata among the partner applications, the Data Warehouse Center, and the Information Catalog Manager.

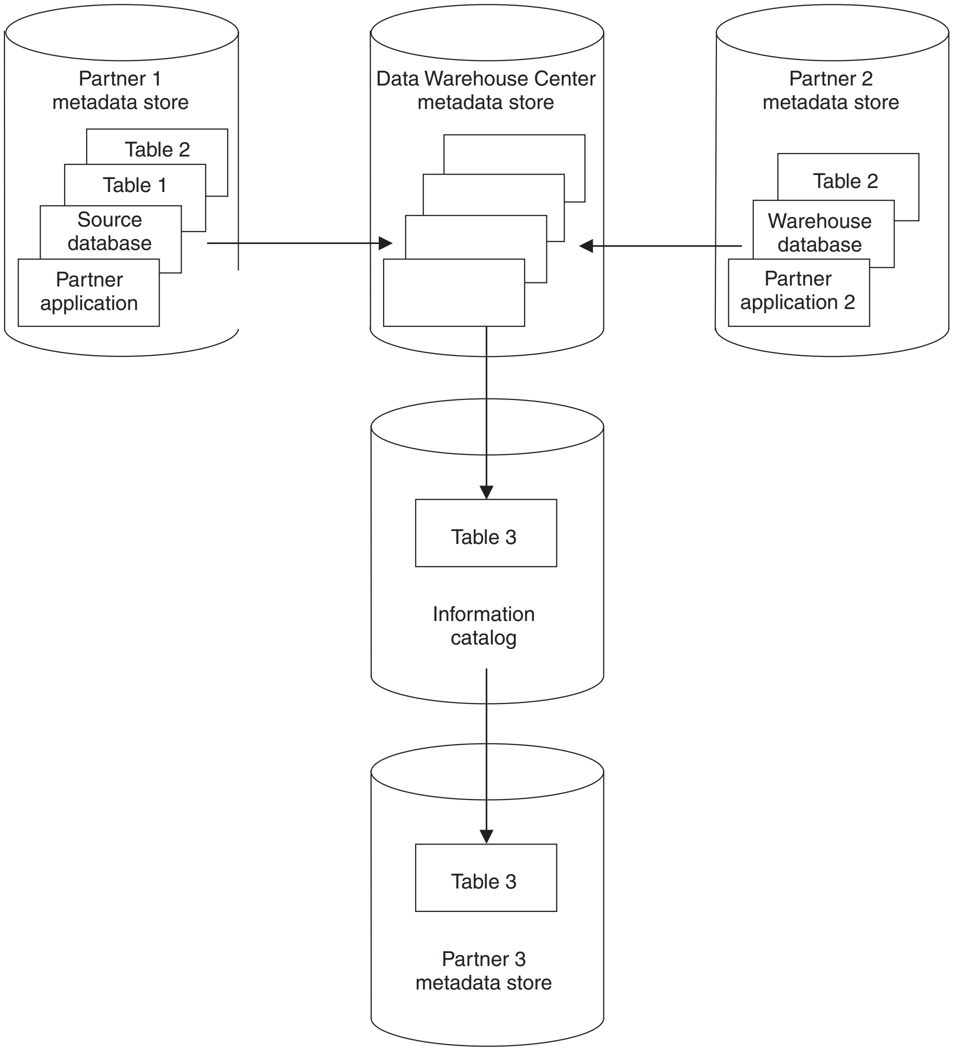


Figure 2. The flow of metadata among the partner applications, the Data Warehouse Center, and the Information Catalog Manager

The rest of this book covers these topics in more detail:

- For more information about importing metadata into the Data Warehouse Center, see “Importing metadata into the Data Warehouse Center” on page 13.
- For more information about exporting metadata from the Data Warehouse Center, see “Exporting metadata from the Data Warehouse Center” on page 35.

- For more information about importing metadata into the Information Catalog Manager, see "Importing metadata into an information catalog" on page 39.
- For more information about exporting metadata from the Information Catalog Manager, see "Exporting metadata from Information Catalog Manager" on page 44.

Integration scenarios

Table 1 lists some common types of warehousing applications and describes how you can integrate them with the Data Warehouse Center.

Table 1. Integration scenarios

Type of application	Integration process
Data warehousing design	To use data from data warehousing design applications in the Data Warehouse Center: 1. Import metadata into the Data Warehouse Center. 2. Use metadata synchronization to propagate metadata into the Information Catalog Manager.
Operational data descriptions	Import metadata into the Data Warehouse Center and business metadata into the Information Catalog Manager. If the metadata is for source data that is included for lineage only and not to define source tables or files, import the metadata into the Information Catalog Manager directly.

Table 1. Integration scenarios (continued)

Type of application	Integration process
Data cleansing	To clean operational data: 1. Determine which application will manage the movement of the source data and target data: the Data Warehouse Center or the partner application. Different applications can manage the source data and the target data. 2. Import source and target definitions, or export source and target definitions, or both. Do so to avoid typing the definitions again. 3. Define the partner application as a Data Warehouse Center program, or write a Data Warehouse Center program that starts the partner application. 4. Develop a user interface that sets the partner application parameters. 5. Import the metadata into the Data Warehouse Center so that the Data Warehouse Center can run the data cleansing application. You can schedule programs by sequence as well as date and time. 6. Import business metadata into the Information Catalog Manager for use by users.
Alternate data storage (such as DB2 OLAP Server™)	To load operational data into alternate data storage: 1. From the Data Warehouse Center, export the data definitions that are needed to build the partner storage. 2. Define the load programs as a Data Warehouse Center program, or write a Data Warehouse Center program that starts the load programs. 3. Develop a user interface that sets the partner application parameters. 4. Import definitions of the load programs into the Data Warehouse Center. Use the load programs to synchronize the values in the operational data store and in the partner data store. 5. Import business metadata for the partner data store into the Information Catalog Manager.

Table 1. Integration scenarios (continued)

Type of application	Integration process
Reporting (such as Brio or Cognos)	To integrate reporting applications with the Data Warehouse Center: 1. Export business metadata from the Information Catalog Manager into the report application. 2. Import descriptions of the reports into the Information Catalog Manager. 3. Enable starting of the report application from an information catalog.

Hardware and software requirements

The models and templates that are described in this book require Data Warehouse Center Version 7.1 which is available in the DB2 Universal Database package, Information Catalog Manager Administrator Version 7.1 which is available in the Warehouse Manager package, and their prerequisite products.

For information about the prerequisite products for the Data Warehouse Center and the Information Catalog Manager, see the *Quick Beginnings* book for your platform and the *DB2 Warehouse Manager Installation Guide*.

Chapter 2. Importing and exporting metadata

This chapter provides detailed information about how to import metadata directly into, and export metadata directly from, the Data Warehouse Center.

Importing metadata into the Data Warehouse Center

You import metadata into the Data Warehouse Center so that the Data Warehouse Center can extract and transform data for the warehouse or run partner applications that extract and transform data.

Importing metadata into the Data Warehouse Center involves the following tasks:

1. Build a *tag language file* (a file that contains the metadata for the objects to import).
2. Import the tag language file.
3. Prepare the steps to run on your data warehouse.

Building the tag language file

To build the tag language file:

1. Select the objects for which to import metadata.
2. Define the metadata for each object by using the Data Warehouse Center metadata templates. The *Data Warehouse Center metadata templates* are subsets of the tag language file that include tokens to represent a partner metadata value. Your program can search for the tokens and substitute values for them without referring to the syntax of the tag language file.

Selecting objects for which to import metadata

You can import metadata for the following types of objects into the Data Warehouse Center:

Agent sites

A *warehouse agent* performs the actual transfer of data between the source database or file (warehouse source), and the target database (warehouse target). It also performs any transformation of that data. The warehouse agent receives commands from the warehouse server. Then, the agent issues SQL commands, starts a partner application, or starts a Data Warehouse Center program that starts a partner application. A warehouse agent can also import table definitions.

An *agent site* is the machine on which an agent runs. The agent site must have access to the machine that contains the source database and the target database.

Warehouse sources and warehouse targets

A *source database* or *source file* is the database or file from which the Data Warehouse Center or a partner application extracts data for further processing. The generic term *source* means a database or a group of one or more files. A source is associated with one or more tables, files or segments. A table, file or segment is associated with one or more columns or fields. A warehouse source is a subset of tables and views from a single database, or a set of files, that have been defined to the Data Warehouse Center.

A *warehouse target* or *target file* is the database or file to which the Data Warehouse Center or a partner application writes the data after processing it. The generic term *target* means a database or a group of one or more files. A target is associated with one or more tables or files. A table or file is associated with one or more columns or fields. A warehouse target is a subset of tables, or a set of files, that are managed by the Data Warehouse Center.

A warehouse target is the database that contains the warehouse that users will use to run queries and reports.

Data Warehouse Center programs

A *Data Warehouse Center program* is a user-written or partner application that performs some kind of data transformation. You define the program to the Data Warehouse Center so that you can schedule it to run and monitor its operations as part of a step. A Data Warehouse Center program is generally associated with one or more parameters. You can group related Data Warehouse Center programs together by associating them with a Data Warehouse Center program group.

Subject areas

You use a *subject area* to logically group the processes (and the steps, warehouse sources, and warehouse targets within the processes) that are related to a particular topic or function. For example, if you have a series of processes that move and transform sales data, you create a Sales subject area, and create the processes within the subject area. Similarly, you group Marketing processes under a Marketing subject area.

Processes

A process is a a series of steps, which commonly operates on source data, that changes data from its original form into a form conducive to decision support. A Data Warehouse Center process commonly consists of one or more warehouse sources, one or more steps, and one or more warehouse targets.

Steps A step is a single operation on data in a Data Warehouse Center process. A process commonly consists of one or more warehouse sources, one or more steps, and one or more warehouse targets. In most cases, a step includes a warehouse source, a description of the transformation or movement of data, and a warehouse target. You use steps to define and schedule each step in the extraction, transformation, and writing of the data. The metadata for a step includes the source and target tables on which the Data Warehouse Center or the partner application is to operate. It also includes the SQL statements to issue or the program to start to perform the transformation.

Cascade relationships between steps
A *cascade relationship* is a schedule for a step that is based on the processing status of another step. You can schedule a step to run after another step finishes running.

Relationships between Data Warehouse Center objects
The metadata for Data Warehouse Center objects describes relationships to other objects. For example, the metadata for a step describes relationships to the warehouse source and warehouse target tables that the step uses.

Defining objects with the Data Warehouse Center metadata templates

To define objects that you want to import into the Data Warehouse Center, you build a tag language file from one or more Data Warehouse Center metadata templates.

Each template corresponds to an object, such as a table, or a subset of an object, such as a column. You combine templates to define all the details about an object. For example, if you want to define a source database, you combine database, table, and column templates.

You must write a program that obtains values from the partner metadata store and use these values to replace tokens in the template. This book calls this type of program an *interchange program*.

Each template contains tokens for which your interchange program must specify values. For example, the token `*TableDescription` represents the description of a table. Your interchange program would search for `*TableDescription` and change it to the string that contains the description of the table specified in the relational catalog. For a DB2™ Universal Database table, the description is in the REMARKS field of the syscat.tables table of the system catalog. Because your interchange program replaces the tokens with a value, you do not need to know the syntax of the underlying tag language that identifies metadata in the file.

Installing the metadata templates: You can choose to install the templates when you install the Application Development Client.

To install the templates:

1. Click **Custom** on the installation Setup Type window.
2. Click **Data Warehouse ISV Toolkit**.
3. Select the directory for the templates.

   The default directory for the ISV Toolkit is x:\sqllib\templates. The Data Warehouse Center sets the *VWS_TEMPLATES* environment variable to the location of the ISV Toolkit. Your program can query the value of *VWS_TEMPLATES* to locate the templates.

The Data Warehouse Center installs the files in subdirectories of the directory that is set by *VWS_TEMPLATES*. Table 2 lists the types of files that are installed and the subdirectories in which the files are installed.

Table 2. File types and subdirectories for templates

Type of file	Subdirectory
Templates	ISV
Samples	Samples
Header files	Include

Writing an interchange program: When you write an interchange program, you need to:

- Include the header file.
- Copy and change the appropriate templates.
- Set checkpoints in each copy of a template.
- Append the changed copies of the templates to the tag language file.

You can also log processing messages in the same directory that the Data Warehouse Center uses to log processing messages.

Including the ISV_defines.h header file: Use of the ISV_Defines.h header file allows your program logic to stay the same even if the template's tokens change. You simply need to recompile your program.

Copying and changing templates: Your program must use the following procedure to work with the templates:

1. Use the *VWS_TEMPLATES* environment variable to obtain the directory in which the templates are stored. Append `\ISV\` to the value to obtain the complete path for the templates.
2. Read a copy of the templates locally into your program.
3. Search the templates for the tokens in the templates and replace the tokens with the metadata from the partner application.

   Use a search and replace methodology, rather than programming to the format of the tag language file. Use of the tokens enables your program to be independent of changes to the tag language that is used in the template file.

   In the templates, each token is enclosed in parentheses; the closing parenthesis identifies the end of the value. Your program should substitute values for only the token and not remove the parentheses.

   Any string that is to replace a token value must follow the following rules:
   - The string must not contain embedded tab characters.
   - Any parenthesis in the string must be enclosed in single quotation marks.

     For example, if you want to replace the `*DatabaseNotes` token with the value:

     ```
     This is my database (managed by the Finance group).
     ```

     You must change the value to:

     ```
     This is my database '('managed by the Finance group')'.
     ```

   If your interchange program does not have a value for a token, it should replace the token with the constant `ISV_DEFAULTVALUE` (defined in ISV_defines.h). However, you must specify a value other than `ISV_DEFAULTVALUE` for any token that is required.

   Because there is no template for security groups, your program must specify the value `ISV_DEFAULTSECURITYGROUP` for any instances of the *SecurityGroup token.

   The templates use default values for Data Warehouse Center specific metadata. For example, retry count and retry interval for warehouse sources and warehouse targets are set to their Data Warehouse Center default values.

Setting checkpoints: Each template contains a *CurrentCheckPointID++ token, which you can use to track progress when you import the tag language file. When your program sets values for the tokens, it should set the first occurrence of *CurrentCheckPointID++ to 0. Your program should increase the value of *CurrentCheckPointID++ by 1 each time it appears. The Data Warehouse Center will write these checkpoints to the log file as the tag language file is being imported.

Appending templates to the tag language file: Tables 3, 4, and 5 list the order in which your program must append templates to the tag language file. They also provide the conditions under which the template is required or optional.

Except for the header, you can define as many copies of each template as you need. You must define only one copy of the header in each tag language file.

Table 3. Relationships between templates and conditions

Order	Template	Required or optional
1	HeaderInfo.tag	Always required
2	AgentSite.tag	Required if you do not use the default agent site
3	VWPGroup.tag	Required if you are defining Data Warehouse Center programs
4	VWPProgramTemplate.tag	Required if you are defining Data Warehouse Center programs
5	VWPProgramTemplateParameter.tag	Required if you are defining Data Warehouse Center programs
6	SourceDataBase.tag WarehouseDataBase.tag	Required if you are defining warehouse sources or warehouse targets
7	Table.tag	Required if you are defining warehouse sources or warehouse targets
8	Column.tag	Required if you are defining warehouse sources or warehouse targets

After you append the Column.tag template to the tag language file, the series of templates and the order in which the templates are appended to the tag language file depend on whether you want to define a step or a star schema.

If you are defining a step, append the following templates to the tag language file in the order shown in Table 4.

Table 4. Relationships between templates and conditions when defining a step

Order	Template	Required or optional
9	SubjectArea.tag	Required if you are defining steps.
10	Process.tag	Required if you are defining steps.

Table 4. Relationships between templates and conditions when defining a step (continued)

11	Step.tag	Required if you are generating SQL transformations between source and target data or defining programs that the Data Warehouse Center is to execute.
12	StepInputTable.tag	Required if you are defining a step of type: ISV_StepType_Editioned_Append ISV_StepType_Full_Replace ISV_StepType_Uneditioned_Append Optional if you are defining a step of type: ISV_StepType_VWP_Population
13	StepOutputTable.tag	Required if you are defining a step of type: ISV_StepType_Editioned_Append ISV_StepType_Full_Replace ISV_StepType_Uneditioned_Append StepOutputTable cannot be used for steps of type: ISV_StepType_VWP_Population
14	StepVWPOutputTable.tag	Optionalif you are defining a step of type: ISV_StepType_VWP_Population
15	StepCascade.tag	Required in order to link steps in a cascaded relationship
16	StepVWPProgramInstance.tag	Required if the step uses a Data Warehouse Center program
17	VWPProgramInstanceParameter.tag	Required if the step uses a Data Warehouse Center program which expects parameters to be passed and has parameters.

If you are defining a star schema, append the following templates to the tag language file in the order shown in Table 5.

Table 5. Relationships between templates and conditions for defining a star schema

Order	**Template**	**Required or optional**
9	StarSchema.tag	Required if you are defining a star schema.
10	StarSchemaInputTable.tag	Required if you are defining a star schema.

For detailed information about these templates, see "Chapter 5. Metadata templates" on page 53.

Logging processing messages: Your interchange program can write log processing messages or trace files to the directory that the *VWS_LOGGING* environment variable specifies. The Data Warehouse Center uses this directory for its log files and its trace files.

Defining the header for the tag language file: To define the objects that a tag language file can contain, you must define the header.

To define the header:
1. Copy the applicable template.
2. Substitute actual values for tokens.

Copying templates: Your program must copy and change the HeaderInfo.tag template file.

Substituting values: Your program must supply the following values:
- The default security group, ISV_DEFAULTSECURITYGROUP
- The value of the *CurrentCheckPointID*++ token for the metadata for the header

For information about the tokens in the template, see "HeaderInfo.tag" on page 63.

Program logic: Figure 3 on page 21 is a pseudocode example of the logic that your program can use to build the header portion of the tag language file.

```
Initialize native metadata environment (need to include ISV_defines.h)
Read a copy of the HeaderInfo.tag template (from the templates directory)
Search for and replace tokens with the metadata from your native metadata
   store (or defaults)
Write the output to a target file
```

Figure 3. Pseudocode for adding the header to the tag language file

The ISV_Sample program provides an example of the header portion of the tag language file. You can find the source code for the program in the Samples subdirectory of the directory that is set by the *VWS_TEMPLATES* environment variable.

Defining agent sites

You can use one of the following agent site types:

- An agent site that is already defined in the warehouse control database.

  To use an existing agent site, replace all occurrences of the **AgentSite* token with the agent site name.
- The default agent site.

  To use the default agent site, replace all occurrences of the **AgentSite* token with `ISV_DEFAULTAGENTSITE`.
- A new agent site that you define using the AgentSite template.

  To define a new agent site, specify values for the tokens in the AgentSite template. Replace all occurrences of the **AgentSite* token with the name of the new agent site.

To define a new agent site:

1. Copy the applicable template.
2. Substitute actual values for tokens.

Copying templates: Your program must copy and change the AgentSite.tag template file. The AgentSite.tag template requires the HeaderInfo.tag template as a prerequisite.

Substituting values: To define a new agent site, your program must obtain metadata about the workstation on which the warehouse agent is installed. Your program must substitute the values that it obtains for the appropriate tokens in the template.

Program logic: Figure 4 on page 22 shows a pseudocode example of the logic your program can use to add a new agent site to the tag language file.

```
If the ISV wants to create an AgentSite specific to the ISV:
     Read a copy of the AgentSite.tag template from the template directory
     Search for and replace tokens with the metadata from your native
       metadata store (or defaults)
     Append the output to a target file
Else
     Set AgentSite token to default agentsite value
```

Figure 4. Pseudocode example of modifying the AgentSite.tag template

The ISV_Sample program provides an example of adding an agent site that is specific to a partner tool to the tag language file. You can find the source code for the program in the Samples subdirectory of the directory that is set by the *VWS_TEMPLATES* environment variable.

Defining sources and targets

You define sources if you want the Data Warehouse Center or a partner application to read data from those sources. Similarly, you define targets if you want the Data Warehouse Center or a partner application to write data to those targets. You must define any sources and targets that are used, except under the following conditions:

- The source or target is already in the warehouse control database.
- You are using only the steps that use Data Warehouse Center programs.

To define sources and targets:

1. Copy the applicable templates.
2. Substitute actual values for tokens.

Copying templates: You can define the following types of source objects:

- Relational databases
- IMS databases
- File systems
- Files

You can define relational databases as target objects.

Tables 6 and 7 list the templates that your program must copy and change to define each type of source and target object.

Relational tables: Table 6 on page 23 lists the templates that your program must copy to define a relational database.

Table 6. Templates for relational source and target definitions

Source or target definition	Number of copies of template	Template to copy	Prerequisite template
Database	One copy for each database you want to use	SourceDataBase.tag (see page 78) WarehouseDataBase.tag (see page 98)	HeaderInfo.tag (see page 63) AgentSite.tag (see page 55) if you are not using the default agent
Table	One copy for each table that you want to define in the database	Table.tag (see page 83)	SourceDataBase.tag (see page 78) WarehouseDataBase.tag (see page 98)
Column	One copy for each column that you want to define in each table	Column.tag (see page 57)	Table.tag (see page 83)

You relate the templates for the tables to the template for the database by specifying common values in the templates. Similarly, you relate templates for the columns to the template for the table by specifying common values in the templates.

Figure 5 shows the relationship between the database, table, and column templates. The 1 to m notation indicates a one to many relationship, where many is inclusive of zero.

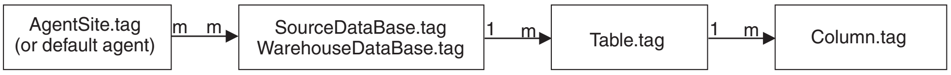


Figure 5. Relationship between the DataBase.tag, Table.tag, and Column.tag templates

IMS databases: Table 7 lists the templates that your program must copy to define an IMS database. You must use the Data Warehouse Center ODBC drivers to access these IMS objects.

Table 7. Templates for IMS source definitions

Source or target definition	Number of copies of template	Template to copy	Prerequisite template
Database	One copy for each database you want to use	SourceDataBase.tag (see page 78)	HeaderInfo.tag (see page 63) AgentSite.tag (see page 55) if you are not using the default agent

Table 7. Templates for IMS source definitions (continued)

Source or target definition	Number of copies of template	Template to copy	Prerequisite template
Segment	One copy for each segment that you want to use in the database	Table.tag (see page 83)	SourceDataBase.tag (see page 78)
Field	One copy for each field that you want to use in each segment	Column.tag (see page 57)	Table.tag (see page 83)

You define relationships between the templates for the database, segments, and fields in the same manner that you define relationships for tables. (See Figure 5 on page 23.)

Files: Table 7 on page 23 lists the templates that your program must copy to define either a file system and its associated files, or a single file.

Table 8. Templates for file systems or a single file

Source or target definition	Number of copies of template	Template to copy	Prerequisite template
File system	One copy for each file system	SourceDataBase.tag (see page 78)	HeaderInfo.tag (see page 63) AgentSite.tag (see page 55) if you are not using the default agent
File	One copy for each file that you want to use in the file system	Table.tag (see page 83)	SourceDataBase.tag (see page 78)
Field	One copy for each field that you want to use in each file	Column.tag (see page 57)	Table.tag (see page 83)

You define relationships between the templates for the file system, files, and fields in the same manner that you define relationships for tables. (See Figure 5 on page 23.)

Substituting values: Your program must obtain values that describe databases or files from the partner metadata store. Your program must substitute the values that it obtains for the appropriate tokens in the template.

Databases: Your program must supply the following metadata about the source databases or the target databases:

- The source databases to define or the target databases to define
- The machines on which the databases reside
- The tables in each database to define
- The columns in each table to define

Files: Your program must supply the following metadata about the source files:

- The file system that contains the files
- The source files to define or target files to define
- The machines on which the files reside
- The fields in each file to define

Program logic: Figure 6 shows a pseudocode example of the logic that your program can use to create or update data resources for source or target definitions.

```
For each source or target to be defined:
   Read a copy of the SourceDatabase.tag or WarehouseDatabase.tag template
   Search for and replace tokens with the metadata from your native metadata source
      (or defaults)
   Append the output to a target file

   For each table, file, or segment that is to be defined:
      Read a copy of the Table.tag template
      Search for and replace tokens with the metadata from your native metadata source
         (or defaults)
      Append the output to a target file

      For each column or field that the table contains:
         Read a copy of the Column.tag template
         Search for and replace tokens with the metadata from your native metadata source
            (or defaults)
         Append the output to a target file
      End (for each column)
   End (for each table)
End (for each source or target data source)
```

Figure 6. Pseudocode for creating or updating data resources for source and target definitions. Use this logic for each source or target definition that you want to create or update.

The ISV_Sample program provides an example of creating or updating data sources for source or target definitions. You can find the source code for the program in the Samples subdirectory of the directory that is set by the *VWS_TEMPLATES* environment variable.

Defining Data Warehouse Center programs

If you want the Data Warehouse Center to schedule and run a partner application, you must first define the application as a Data Warehouse Center program. Then you can schedule and run the program by using it in one or more steps.

If your tag language file is to contain Data Warehouse Center programs, you must define the following objects, in order:

1. One or more program groups to contain the Data Warehouse Center programs.
2. One or more Data Warehouse Center program templates, which provide the base definition of the program to the Data Warehouse Center.
3. One or more Data Warehouse Center program template parameters, which provide the default parameters that the Data Warehouse Center passes to the program.

   You can change the parameters that are used in a particular step by defining an instance of the program parameters for the step. For more information about using a Data Warehouse Center program in a step, see "Defining steps" on page 28.

For information about writing a program for use with the Data Warehouse Center, see "Appendix C. Writing your own program to use with the Data Warehouse Center" on page 295.

To define a Data Warehouse Center program:

1. Copy the applicable template.
2. Substitute actual values for tokens.

Copying templates: Table 9 lists the templates that your program must copy and change to define Data Warehouse Center programs.

Table 9. Templates for Data Warehouse Center programs

Definition	Number of copies of template	Template to copy	Prerequisite template
Data Warehouse Center program group	One copy for each program group to define	VWPGroup.tag (see page 89)	HeaderInfo.tag (see page 63)

Table 9. Templates for Data Warehouse Center programs (continued)

Definition	Number of copies of template	Template to copy	Prerequisite template
Data Warehouse Center program template	One copy for each Data Warehouse Center program in the program group	VWPProgramTemplate.tag (see page 93)	VWPGroup.tag (see page 89)
Data Warehouse Center program template parameter	One copy for each parameter passed to theData Warehouse Center program	VWPProgramTemplateParameter.tag (see page 95)	VWPProgramTemplate.tag (see page 93)

You relate the templates for the Data Warehouse Center program group to the template for the Data Warehouse Center program by specifying common values in the templates. Similarly, you relate templates for the parameters to the template for the Data Warehouse Center program by specifying common values in the templates.

Figure 7 shows the relationship between the Data Warehouse Center program group, the Data Warehouse Center program, and the Data Warehouse Center program parameters.

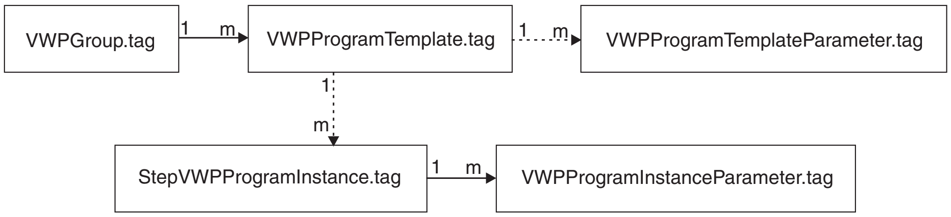

VWPGroup.tag
1
m
VWPProgramTemplate.tag
1
m
VWPProgramTemplateParameter.tag
1
m
StepVWPProgramInstance.tag
1
m
VWPProgramInstanceParameter.tag

Figure 7. Relationship between the VWPGroup.tag, VWPProgramTemplate.tag, and VWPProgramTemplateParameter.tag templates

For information about relating a Data Warehouse Center program to a step, see “Defining steps” on page 28.

Substituting values: Your program must obtain values that describe the Data Warehouse Center programs from the partner metadata store:

- The Data Warehouse Center program groups to define
- The Data Warehouse Center programs to define
- The parameters in each Data Warehouse Center program to define

Your program must substitute the values that it obtains for the appropriate tokens in the templates.

Program logic: Figure 8 shows a pseudocode example of the logic that your program can use to define applications that will be managed and run by the Data Warehouse Center.

```
Read a copy of the VWPGroup.tag template
Search for and replace tokens with the metadata from your native metadata store
     (or defaults)
Append the output to a target file

For each application that is to be managed by the Data Warehouse Center:
     Read a copy of the VWPProgramTemplate.tag template
     Search for and replace tokens with the metadata from your native metadata store
          (or defaults)
     Append the output to a target file

     For each parameter the application needs passed:
          Read a copy of the VWPProgramTemplateParameter.tag template
          Search for and replace tokens with the metadata from your native metadata store
               (or defaults)
          Append the output to a target file
     End (for each parameter)
End (for each application)
```

Figure 8. Pseudocode for defining Data Warehouse Center programs

The ISV_Sample program provides an example of adding Data Warehouse Center programs to the tag language file. You can find the source code for the program in the Samples subdirectory of the directory that is set by the *VWS_TEMPLATES* environment variable.

Defining steps

A step is a single operation on data in a warehouse process. In most cases, a step includes a warehouse source, a transformation or movement of data, and a warehouse target. A step can be run according to a schedule, or it can cascade from another step. You use steps to define and schedule each step in the extraction, transformation, and writing of the data. You must define a step for each part of the transformation process that you want the Data Warehouse Center to manage. Use the information in this section to determine how to define your steps, rather than the information in the Data Warehouse Center online help. The templates require different relationships from steps that are defined using the user interface.

You must define a subject area for the steps. You can use subject areas to group steps that use a particular partner application.

If your tag language file contains steps, you must define the following objects, in order:

1. One or more subject areas to contain the proceses.
2. One or more processes to contain steps.
3. One or more steps.
4. For each step, a relationship to one or more source tables and a target table if the step uses SQL to do the source-target mapping. If the step uses a Data Warehouse Center program, the source tables and target table are optional.
5. If the step uses a Data Warehouse Center program:
   a. An instance of the Data Warehouse Center program.
   b. The parameters associated with the Data Warehouse Center program.
   c. Optionally, the output table for the Data Warehouse Center program.

To define steps:

1. Copy the applicable template.
2. Substitute actual values for tokens.

Copying templates: Table 10 lists the templates that your program must copy and change to define steps.

Table 10. Templates for steps

Definition	**Number of copies of template**	**Template to copy**	**Prerequisite template**
Subject area	One copy for each subject area	SubjectArea.tag (see page 82)	HeaderInfo.tag (see page 63) AgentSite.tag (see page 55) if you are not using the default agent
Process	One copy for each process	Process.tag (see page 63)	SubjectArea.tag(see page 82)
Step	One copy for each step	Step.tag (see page 68)	SubjectArea.tag (see page 82) Process.tag (see page 63)
Source table for the step	One copy for each source table for the step	StepInputTable.tag (see page 73)	Table.tag (see page 83) Step.tag (see page 68) Process.tag (see page 63)

Table 10. Templates for steps (continued)

Definition	Number of copies of template	Template to copy	Prerequisite template
Target table for the step	One copy if the step has a target table	StepOutputTable.tag (see page 74)	Table.tag (see page 83) Step.tag (see page 68) Process.tag (see page 63)
Target table for a step which uses a Data Warehouse Center program	One copy to document each target table updated by the program	StepOutputTable.tag(see page 74)	Table.tag (see page 83) Step.tag (see page 68)
Data Warehouse Center program instance	One copy if the step uses a Data Warehouse Center program	StepVWPProgramInstance.tag(see page 77)	VWPProgramTemplate.tag (see page 93) Step.tag (see page 68)
Data Warehouse Center program instance parameters	One copy for each parameter used in the step	VWPProgramInstanceParameter.tag (see page 90)	StepVWPProgramInstance.tag(see page 77)

You relate the templates for the subject area to the templates for the process by specifying common values in the templates. Similarly, you relate templates for the steps to the templates for input tables and output tables by specifying common values in the templates. You can also relate the template for the step to a template for the program instance by specifying common values in the templates.

Figure 9 on page 31 shows the relationship between the subject area, step, stepinput table, stepoutput table, stepVWP program instance, and the VWP program instance parameter tags.

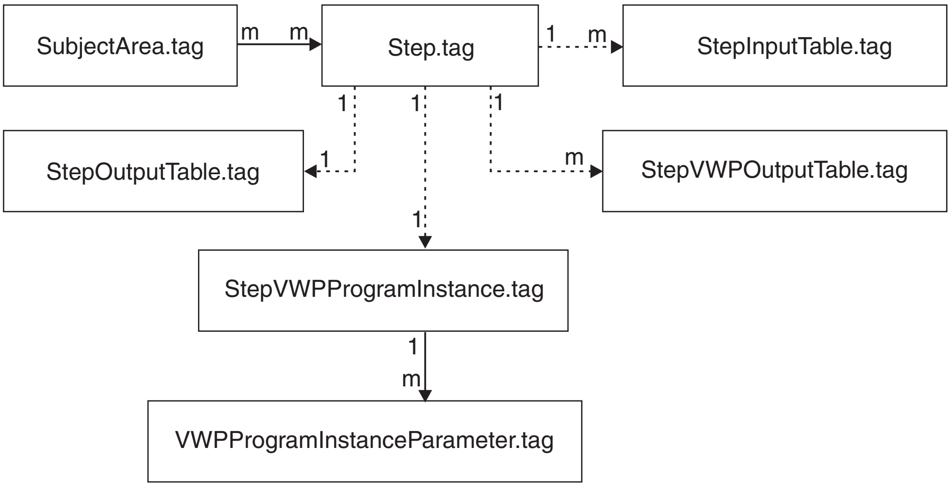


Figure 9. Relationship between the SubjectArea.tag, Process.tag, Step.tag, StepInputTable.tag, StepOutputTable.tag, StepVWPOutputTable.tag,StepVWPProgramInstance.tag, and VWPProgramInstanceParameter.tag templates. See Figure 7 on page 27 to see how the Data Warehouse Center program instance templates relate to the other Data Warehouse Center program templates.

Substituting values: Your program must obtain values that describe the subject areas and steps from the partner metadata store:

- The subject areas to contain the process which contains the steps
- The steps to define
- The source tables for each step
- The target table for each step, if applicable
- The Data Warehouse Center program and parameters for the step, if applicable

Your program must substitute the values that it obtains for the appropriate tokens in the templates.

Program logic: Figure 10 on page 32 shows pseudocode of the logic that your program can use to define steps in the tag language file.

```
Read a copy of the SubjectArea.tag template
Search for and replace tokens with the metadata from your native metadata store (or defaults)
Append the output to a target file
Read a copy of the process

For each step to be defined:
     Read a copy of the Step.tag template
     Search for and replace tokens with the metadata from your native metadata store
      (or defaults)
     Append the output to a target file
     If the step is to execute your application:
          Read a copy of the StepVWPProgramInstance.tag template
          Search for and replace tokens with the metadata from your native metadata store
               (or defaults)
          Append the output to a target file
          For each parameter that your application needs:
               Read a copy of the VWPProgramInstanceParameter.tag template
               Search for and replace tokens with the metadata from your native metadata store
                    (or defaults)
               Append the output to a target file
          End (for each parameter)

          If the step is to be related to its VWP output target data:
               Read a copy of the StepVWPOutputTable.tag template
               Search for and replace tokens with the metadata from your native metadata store
                    (or defaults)
               Append the output to a target file
          End (step relation to its output)
     End (if step to execute your application)

     If the step is to be related to its input source data:
          Read a copy of the StepInputTable.tag template
          Search for and replace tokens with the metadata from your native metadata store
           (or defaults)
          Append the output to a target file
     End (step relation to its source)
     If the step is to be related to its output target data:
          Read a copy of the StepOutputTable.tag template
          Search for and replace tokens with the metadata from your native metadata store
           (or defaults)
          Append the output to a target file
     End (step relation to its target)
End (for each step)
```

Figure 10. Pseudocode for defining steps in the tag language file

The ISV_Sample program provides an example of adding steps to the tag language file. You can find the source code for the program in the Samples subdirectory of the directory that is set by the *VWS_TEMPLATES* environment variable.

Defining cascading steps

In your tag language file, you can specify that steps start other steps:

- You can specify that one step starts after another step successfully finishes processing by defining a post-processing cascade relationship.

To define the cascading steps:

1. Copy the applicable template.
2. Substitute actual values for tokens.

Copying templates: Table 11 lists the templates that your program must copy and change to define cascade relationships.

Table 11. Templates for cascade relationships

Definition	Number of copies of template	Template to copy	Prerequisite template
Step cascade relationship	One copy for each relationship	"StepCascade.tag" on page 72	"StepCascade.tag" on page 72

Substituting values: Your program must supply the name of a step and the name of another step to:

- Start after the first step.

Your program must substitute the values it obtains for the appropriate tokens in the templates.

Program logic: Figure 11 shows pseudocode of the logic that your program can use if you want your application to relate two steps together so that one step starts at the completion of another step.

```
Read a copy of the StepCascade.tag template
Search for and replace tokens with the metadata from your native metadata store
     (or defaults)
Append the output to a target file
End (relate steps for cascaded processing)
```

Figure 11. Pseudocode for relating steps for cascaded processing

The ISV_Sample program provides an example of how to relate steps for cascaded processing in the tag language file. You can find the source code for the program in the Samples subdirectory of the directory that is set by the *VWS_TEMPLATES* environment variable.

Importing metadata from the tag language file

You can import metadata from the tag language file by using a command window or the user interface. This section describes how to use the command window. For information about using the user interface, see the Data Warehouse Center online help.

To import a tag language file, enter the following command at a DOS command prompt:

```
iwh2imp2 tag-filename log-pathname target-control-db userid password
   [PREFIX = schema]
```

tag-filename
: The full path and file name of the tag language file.

log-pathname
: The fully qualified path name of the log file.

target-control-db
: The name of the warehouse control database that is the target database for the import.

userid
: The user ID to use to access the warehouse control database.

password
: The password to use to access the warehouse control database.

[PREFIX = schema]
: The table qualifier for the metadata tables.

  If a prefix is not specified, the default value is *IWH*.

To get help for the import command parameters, enter the command only.

When the import utility imports metadata from a tag language file, it creates a log file with:

- The same file name as the tag language file.
- A file extension of LOG.

The import process records the return code and the last completed checkpoint at the end of the log file.

You can also code the return code into your interchange program by using the system() call or the rexec() call. The call to use depends on the operating system on which your program is running.

For more information about importing metadata into the Data Warehouse Center, see the *Data Warehouse Center Administration Guide*.

Preparing the steps to run

After you import the metadata into the Data Warehouse Center, you must complete the following procedure to set up an automated process for your warehouse:

1. Specify passwords for the following objects:
   - Any agent sites that you imported

- Any warehouse sources or warehouse targets (sources and targets) that you imported

2. For SQL steps, if the source tables or files map directly to the target table, map the source columns to the target columns.
3. After the objects are created in the Data Warehouse Center, define specific date and time schedules for the steps using the Data Warehouse Center. You can also define cascade relationships if you did not do so in the tag language file.
4. Promote the steps to test mode.
5. To test the steps, run them by selecting them in the Run New Step window.

   If you need to make changes:

   a. Demote the steps to development mode if necessary.
   b. Make the changes.
   c. Promote the steps to test mode again.

   Be sure to update your program to account for these changes.
6. Promote the steps to production mode to activate their schedules.

   Your steps will now run on an automated schedule.

Exporting metadata from the Data Warehouse Center

You export metadata from the Data Warehouse Center if you want your partner application to operate on data sources or targets that are defined in the Data Warehouse Center.

Exporting metadata from the Data Warehouse Center involves the following procedures:

1. Select the objects for which to export metadata.
2. Export the metadata to a tag language file.

Selecting objects for which to export metadata

Most Data Warehouse Center objects are specific to the Data Warehouse Center. However, you can use metadata about databases, tables, and columns to define source and target databases for partner applications. You can use this capability to share source and target information between partner applications that transform data for the same warehouse.

For example, one partner tool might unload data from a database into a target file. Another partner tool might use the file as a source file and:

- Read data from that file.
- Transform the data.
- Write the data to another data file.

A third partner tool might read the data from the file and load it into a target database. If you export the metadata for the databases and files from the Data Warehouse Center, you can make sure that all the partner tools are using the same data definitions.

To define source databases, export one or more warehouse sources (all tables and columns are included automatically). To define a target database, export a warehouse target (all tables and columns are included automatically).

When you export the objects, the Data Warehouse Center writes the objects in a file, using tag language format. For information about the tags that are used to identify metadata for source databases and target databases, see "Chapter 6. Data Warehouse Center metadata" on page 103. For the syntax and structure of a tag language file, see "Chapter 9. Tag language" on page 215 and "Chapter 10. What a tag language file should look like" on page 249.

Table 12 shows the mapping between the logical Data Warehouse Center objects and the tag language object that represents the logical object.

Table 12. Logical objects for source and target databases

Data Warehouse Center logical object	Object in tag language file	Description	See:
Warehouse Source	DATABASE	Source database or file	"DATABASE object" on page 103
Warehouse Target	DATABASE	Target database or file	"DATABASE object" on page 103
Table	TABLES	Table, file, or segment in source or target database	"TABLES object" on page 108
Column	COLUMN	Column or field in table or field in file	"COLUMN object" on page 114

Exporting metadata into a tag language file

You can use the Data Warehouse Center user interface or a command window to export metadata from the Data Warehouse Center. This section covers how to use the command window. For information about using the user interface, see the Data Warehouse Center online help and the *Data Warehouse Center Administration Guide*.

First, you create an .INP file with the list of warehouse sources and warehouse targets that you want to export. For example:

```
<IR>
LOG_STAT_IR
LOG_STAT_REP
```

LOG_STAT_IR is a warehouse source, and LOG_STAT_REP is a warehouse target. The Data Warehouse Center automatically exports the tables and columns that are associated with LOG_STAT_IR and LOG_STAT_REP.

Then, to export the tag language file, enter the following command at a DOS command prompt:

```
iwh2exp2 INPfilename controlDBname userid password [PREFIX = schema]
```

INPfilename
: The full path and file name of the .INP file.

  Create this file in a read/write directory because the Data Warehouse Center will write the tag language file in this directory. The Data Warehouse Center names the tag language file *INPfilename*.TAG.

controlDBname
: The name of the control database.

userID
: The user ID required to access the control database.

password
: The password that is required to access the control database.

[PREFIX = schema]
: The table qualifier for the metadata tables.

  If a prefix is not specified, the default value is *IWH*.

For more information about exporting metadata from the Data Warehouse Center, see the *Data Warehouse Center Administration Guide*.

The import formats and the export formats are release-dependent. You cannot use exported files from a previous release to migrate from one release of the Data Warehouse Center to another. For more information on migration, see the *Quick Beginnings* book for your platform.

Chapter 3. Importing and exporting metadata with the Information Catalog Manager

This chapter provides detailed information about how to import metadata directly into, and export metadata directly from, the Information Catalog Manager.

Importing metadata into an information catalog

You can import metadata from the Data Warehouse Center to provide information about the data in a warehouse for the users of the warehouse. You can import metadata from partner applications that also provide some cataloging facilities.

Import metadata into the Information Catalog Manager involves the following tasks:

1. Select the types of metadata to import.
2. Import the metadata into the Information Catalog Manager.

Selecting metadata to import

When you import metadata into an information catalog, you can import the tag language in two formats:

- A format that is used by both the Information Catalog Manager and the Data Warehouse Center
- A format that conforms to MDIS.

Importing metadata from a tag language file

You can import metadata from tag language files that are in MDIS format or in the format used by the Information Catalog Manager and the Data Warehouse Center. See "Chapter 8. Information Catalog Manager object types" on page 147 for mappings of the Information Catalog Manager object types to MDIS names. For more information on MDIS tag language format, visit the Meta Data Coalition's Web site at http://www.MDCinfo.com.

If you are using MDIS with other products and Visual Warehouse 3.1, see the note in "Exporting tag language files" on page 45.

If you want to convert MDIS tag language into an Information Catalog Manager tag language file, see *Information Catalog Manager Administration Guide*.

Importing MDIS-conforming tag language files

To import an MDIS tag language file directly into your information catalog, enter the Information Catalog Manager command from a an MS-DOS command prompt. Adhere to the following rules for the command syntax:

- All the parts, except where specified, are case-insensitive.
- Either a slash (/) or a hyphen must precede each keyword (-).
- All keywords that follow the DGUIDE command are required. All keywords that follow the /MDIS_IMPORT keyword are required.
- Underlined choices are defaults.

DGUIDE /USERID *userid* **/PASSWORD** *password* **/DGNAME** *dgname* **/MDIS_IMPORT** *filename* **/LOGFILE** *filename name***/ADMIN**

Optional keywords:

/TRACE 0|1|2|3|4

For example, to import MDIS metadata into your information catalog, type the following command (do not enter a line break):

```
DGUIDE /USERID longods /PASSWORD secret /DGNAME ICMSAMP /ADMIN
/MDIS_IMPORT c:\mdis.tag /LOGFILE c:\mdis.log
```

/ADMIN
Specifies that you are logging on as an administrator. You must log on as an administrator to import metadata.

/DGNAME
Your information catalog name.

If the information catalog is local, specify the database name. If the information catalog is remote, specify the alias under which it was cataloged.

Example:

```
/DGNAME ICMSAMP
```

/LOGFILE
This parameter is required.

Specifies the file destination for messages that the Information Catalog Manager generates during MDIS import or MDIS export. Unless you specify a full drive, path, and file name, the Information Catalog Manager places the file in the path specified on the DGWPATH environment variable. You must specify a fixed drive.

Example:

```
/LOGFILE d:\tagfile.log
```

/MDIS_IMPORT
Imports the MDIS-conforming tag language file that you specify. Unless

you specify the full drive, path, and file name, the Information Catalog Manager assumes that the file is in the path specified on the DGWPATH environment variable.

Example:

```
/MDIS_IMPORT d:\tagfile.tag
```

The information catalog into which you import MDIS metadata must include, but is not limited to, valid MDIS object type definitions.

/PASSWORD
Your password for this user ID.

Example:

```
/PASSWORD secret
```

Passwords are case-sensitive for accessing databases on the following operating systems, you must type them exactly as specified.

- AIX
- Windows NT and Windows 2000
- Solaris Operating Environment

/TRACE
The level of trace information to send to the Information Catalog Manager trace file. Each higher level includes the functions of the levels below it (for example 3 includes the functions of levels 0, 1, 2, and 3). You might need to specify a higher level if you call IBM Software Support to diagnose the Information Catalog Manager problems.

0	The default. Includes all messages and warning, error, and severe error conditions.
1	Includes entry and exit records of the highest level Information Catalog Manager functions.
2	Includes extremely granular entry and exit records of the Information Catalog Manager functions.
3	Includes input and output parameters (that exclude input or output structures).
4	Includes all input or output structures that are passed to and used by the Information Catalog Manager.

/USERID
Your information catalog user ID. Type the user ID required by the database where the information catalog resides. For example, the user ID might be your local, LAN, AS/400, AIX, or OS/390 TSO user ID.

Example:

```
/USERID longods
```

Importing a tag language file from the command line

Use the DGUIDE command to open an information catalog and import a tag language file from an MS-DOS command prompt. When you use the DGUIDE command, keep in mind the following rules for the command syntax:

- None of the parts, except where specified, are case sensitive.
- Each keyword must be preceded by either a slash (/) or hyphen (-) character.
- All keywords that follow /IMPORT as shown in Figure 12 are required if you choose to import.
- Underlined choices are defaults.

```
DGUIDE /USERID userid /PASSWORD password /DGNAME dgname
```

Optional keywords:

```
/ADMIN
/TRACE 0|1|2|3|4
/IMPORT filename /LOGFILE filename /RESTART B|C
```

Optional import keyword:

```
/ICOPATH iconpath
```

Figure 12. DGUIDE command parameters for opening an î and importing metadata

The following example shows the required parameters you specify to open the sample information catalog as an administrator.

```
DGUIDE /USERID longods /PASSWORD secret /DGNAME ICMSAMP /ADMIN
```

The following list shows the parameters you can add to the DGUIDE command. Optional and required keywords for importing a tag language file are noted.

/ADMIN
Specifies that you are logging on as an administrator. If you do not specify this optional keyword for the DGUIDE command, you are logged on as a user, and you cannot perform administrator tasks.

/DGNAME
Your information catalog name.

If the information catalog is local, give the database name. If the information catalog is remote, give the alias under which it was cataloged.

Example:

```
/DGNAME ICMSAMP
```

/ICOPATH
Valid only with /IMPORT; optional.

Indicates that you are importing icons and specifies the icon path that the import function will use. The Information Catalog Manager assumes that the path is the same as the one where you installed the Information Catalog Manager unless you specify a full drive and path. You must specify a fixed drive.

Example:

```
/ICOPATH d:\icons\
```

/IMPORT
Imports the tag language file you specify. Unless you specify the full drive, path, and file name, the Information Catalog Manager assumes that the file is in the path specified on the DGWPATH environment variable.

Example:

```
/IMPORT d:\tagfile.tag
```

This keyword bypasses the Information Catalog Manager user interface and performs the import function as a batch process.

/LOGFILE
Valid only with /IMPORT; required with /IMPORT.

Specifies the file destination for messages the Information Catalog Manager generates during import. Unless you specify a full drive, path, and file name, the Information Catalog Manager places the file in the path specified on the DGWPATH environment variable. You must specify a fixed drive.

Example:

```
/LOGFILE d:\tagfile.log
```

/PASSWORD
Your password for this user ID.

Example:

```
/PASSWORD secret
```

Passwords are case-sensitive for accessing databases on the following operating systems, you must type them exactly as specified.

- AIX
- Windows NT and Windows 2000
- Solaris Operating Environment

/RESTART
Valid only with /IMPORT; required with /IMPORT.

Indicates which option the import function uses. The valid options are:

B	Imports the tag language file from the beginning.
C	The default. Imports the tag language file from the last point at which the Information Catalog Manager successfully committed changes to the information catalog.

/TRACE
The level of trace information to send to the trace file. Each higher level includes the functions of the levels below it (3 includes the functions of levels 0, 1, 2, and 3). You might have to specify a higher level if you call IBM Software Support to diagnose Information Catalog Manager problems.

0	The default. Includes all messages and warning, error, and severe error conditions.
1	Includes entry and exit records of the highest level Information Catalog Manager functions.
2	Includes extremely granular entry and exit records of the Information Catalog Manager functions.
3	Includes input and output parameters (excluding input or output structure).
4	Includes all input or output structures that are passed to and used by the Information Catalog Manager.

/USERID
Your information catalog user ID. Depending on the database location of the information catalog you are opening, type the user ID required by the database. For example, the user ID might be your local, LAN, AS/400, AIX, or OS/390 TSO user ID.

Example:

```
/USERID longods
```

Exporting metadata from Information Catalog Manager

You can export metadata from the Information Catalog Manager for use by partner applications. For example, you can export Information Catalog Manager metadata for use by a CASE tool that application developers use to develop applications for the data warehouse.

Exporting metadata from the Information Catalog Manager involves the following tasks:

1. Select the types of metadata to export.
2. Export the metadata from the Information Catalog Manager.

When you export metadata from the Information Catalog Manager, you can generate tag language in two formats. For example:

- If you export using the Information Catalog Manager product windows or the FLGExport API, the tag language generated is in the Information Catalog Manager tag language format. You can export metadata from the Windows® 95, Windows NT®, or Windows 2000 command line. For more information, see the *Information Catalog Manager Administration Guide*.
- If you export using the FLGMdisExport API, the tag language generated is in MDIS format. For more information on Information Catalog Manager APIs, see the *Information Catalog Manager Programming Guide and Reference.*

Selecting metadata to export

The metadata that you can export from the Information Catalog Manager is in the form of object types. An *object type* is a classification for objects that is used to reflect a type of business information, such as a table, report, or image.

An information catalog can contain predefined object types and object types that the information catalog administrators define. Predefined object types are object types whose definitions are shipped with the Information Catalog Manager. See "Chapter 8. Information Catalog Manager object types" on page 147 for a description of those object types.

For information on creating object types using the Information Catalog Manager product windows or tag language, see the *Information Catalog Manager Administration Guide.*

Exporting tag language files

Note to those currently using MDIS with other products and Visual Warehouse 3.1: If you already had MDIS configuration and profile files, the Visual Warehouse installation program did not overwrite them. However, before you use the MDIS function of the Information Catalog Manager for the first time, you must merge the information in the Information Catalog Manager MDIS profile and configuration files with your existing files. Complete the following steps:

1. Check the MDIS environment variable setting to locate your existing MDIS profile file (MDISTOOL.PRO) and configuration file (MDISTOOL.CFG).
2. Using a text editor, append the contents of X:\VWSLIB\METADATA\PROFILES\MDISTOOL.PRO to your existing profile file. (X is the drive where you installed the Information Catalog Manager.)
3. Using a text editor, append the contents of X:\VWSLIB\METADATA\PROFILES\MDISTOOL.CFG to your existing configuration file. (X is the drive where you installed the Information Catalog Manager.)

Exporting MDIS-conforming tag language files

To export an MDIS tag language file directly from your information catalog, enter the DGUIDE command from an MS-DOS command prompt. Adhere to the following rules for the command syntax:

- All the parts, except where specified, are case-insensitive.
- Either a slash (/) or a hyphen must precede each keyword (-).
- All keywords that follow the DGUIDE command are required. All keywords that follow the /MDIS_EXPORT keyword are required.

DGUIDE /USERID *userid* **/PASSWORD** *password* **/DGNAME** *dgname* **/MDIS_EXPORT** *filename* **/LOGFILE** *filename* **/OBJTYPE** *object_type* **/OBJECTS** *name*

Optional keywords:

/ADMIN
/TRACE 0|1|2|3|4

For example, to export MDIS metadata from your information catalog to a file, type the following command (do not enter line breaks):

```
DGUIDE /USERID longods /PASSWORD secret /DGNAME ICMSAMP /ADMIN
/MDIS_EXPORT c:\mdis.tag /LOGFILE c:\mdis.log
/OBJTYPE database /OBJECTS server01.payroll.valdezma
```

/ADMIN
Specifies that you are logging on as an administrator. If you do not specify this optional keyword for the DGUIDE command, you log on as a user. You can export metadata as a user; however, you cannot perform all administrator tasks.

/DGNAME
Your information catalog name.

If the information catalog is local, specify the database name. If the information catalog is remote, specify the alias under which it was cataloged.

Example:

```
/DGNAME ICMSAMP
```

/LOGFILE
Specifies the file destination for messages that the Information Catalog Manager generates during MDIS import or MDIS export.

Unless you specify a full drive, path, and file name, the Information Catalog Manager places the file in the path specified on the DGWPATH environment variable. You must specify a fixed drive.

Example:

```
/LOGFILE d:\tagfile.log
```

/MDIS_EXPORT

Exports MDIS-conforming metadata into an MDIS-conforming tag language file with the name that you specify. Unless you specify the full drive, path, and file name, the Information Catalog Manager places the file in the path specified on the DGWPATH environment variable.

Example:

```
/MDIS_EXPORT d:\tagfile.tag
```

The information catalog from which you export MDIS metadata can contain metadata other than MDIS metadata, but /MDIS_EXPORT exports only metadata that conforms to MDIS.

/OBJECTS

This parameter is required.

Specifies the objects you want to export. Depending on the object type that you specified on the /OBJTYPE keyword, the *name* value is from three to five property values, separated by periods.

/OBJTYPE	/OBJECTS
Database	*ServerName.DatabaseName.OwnerName*
Dimension	*ServerName.DatabaseName.OwnerName.DimensionName*
Subschema	*ServerName.DatabaseName.OwnerName.SubschemaName*
Record	*ServerName.DatabaseName.OwnerName.RecordName*
Element	*ServerName.DatabaseName.OwnerName.RecordName.ElementName*

In this list, the parts of the name are represented with their MDIS name. To find the equivalent information catalog names, see *Data Warehouse Center Application Integration Guide*, available from the Data Warehouse Center Web site at http://www.software.ibm.com/data/vw/.

/OBJTYPE

This is a required parameter.

Specifies one of the following MDIS object types that you want to export:
Database
Dimension
Subschema
Record
Element

The object type name is not case sensitive.

Example:

```
/MDIS_EXPORT d:\tagfile.tag /OBJTYPE record
```

/PASSWORD

Your password for this user ID.

Example:

```
/PASSWORD secret
```

Passwords are case-sensitive for accessing databases on the following operating systems, you must type them exactly as specified.

- AIX
- Windows NT and Windows 2000
- Solaris Operating Environment

/TRACE

The level of trace information to send to the Information Catalog Manager trace file. Each higher level includes the functions of the levels below it (for example 3 includes the functions of levels 0, 1, 2, and 3). You might need to specify a higher level if you call IBM Software Support to diagnose the Information Catalog Manager problems.

0	The default. Level includes all messages and warning, error, and severe error conditions.
1	Includes entry and exit records of the highest level Information Catalog Manager functions.
2	Includes extremely granular entry and exit records of the Information Catalog Manager functions.
3	Includes input and output parameters (that exclude input or output structures).
4	Includes all input or output structures that are passed to and used by the Information Catalog Manager.

/USERID

Your information catalog user ID. Type the user ID required by the database where the information catalog resides. For example, the user ID might be your local, LAN, AS/400, AIX, or OS/390 TSO user ID.

Example:

```
/USERID longods
```

Chapter 4. Ensuring that users can start programs from the Information Catalog Manager

You set up the objects in your information catalog so that your users can run application programs to work with the actual information that the objects describe. Users can run the application programs that they are familiar with, including the programs that were originally used to create the information.

Ensure that the following requirements are met:

- Your users need the appropriate application software installed on their workstations or on the LAN.
- Users can launch any program that can be started from the command line with the command `start program_name` without a path, regardless of where the program is installed.

  Many programs write their path to the program registry when they are installed. The `start` command retrieves the path. If a program does not write its path to the program registry, you might need to add the directory path of the program to the path environment variable on users' workstations.
- Your users need the necessary authorization to the databases or file systems where the information that they need is stored.
- The Programs objects in the information catalog must include the correct invocation syntax for the operating systems on which your users will run the programs.

Additional requirements for Information Catalog Manager for the Web users

When you set up the Web environment for Information Catalog Manager for the Web users, ensure that the following requirements are met:

- The data that users want to use with the application program must be accessible to the Web server. For example, the Information Catalog Manager sample data file is located in a directory on the Web server.
- The program that users want to start must be installed on the Web client.

  For example, if users are accessing a Lotus® 1-2-3 file®, then Lotus 1-2-3 must be installed on the Web client.

  If the application program is a Java™ applet, the application does not need to be installed; it can be accessed directly from the Web browser.

  The client should also have any necessary browser plug-in programs. The Information Catalog Manager for the Web server must be able to locate any associated files that are used by the plug-in program. For example, if the users want to view Adobe Acrobat files, they need the browser plug-in program for the Acrobat Reader installed on the Information Catalog Manager for the Web client. The Information Catalog Manager for the Web server must be able to locate the file that the user wants to view to download it to the client.

- The required MIME types must be identified in the Web server configuration file for the application program that users will start. An AddType directive with the file extension of the program that users want to start must be included in the configuration file. For example, if users want to use Lotus 1-2-3 spreadsheets with a file type of WK4, define the AddType directive for Lotus Domino™ Go Webserver as shown in this example:

  ```
  AddType .WK4 application/x-lotus1-2-3 binary
  ```

  If users are using a Web server other than Lotus Domino Go Webserver, the MIME types are defined differently. See your Web server documentation for more information.
- If you are using Websphere IBM HTTP WebServer, the MIME types are defined in the \conf\mime.types file as shown in this example:

  ```
  application/vnd.lotus-1-2-3 wks 123 wk1 wk2 wk3 wk4
  ```
- For some versions of Netscape Navigator, helper programs recognize file types and start the corresponding application program. Microsoft Internet Explorer does not use helper programs. Instead, Internet Explorer uses file type and program associations that are used by Windows Explorer; no setup is required for Internet Explorer to recognize a file type.
- The **URL to access data** property must be defined for the object from which users want to start the program. The value for the property is a link to directly launch the program.

To start a program from an Information Catalog Manager for the Web object:

1. In the list pane, click on the object from which you want to start the program.

   The object description page opens in the description pane.
2. Find the **URL to access data** property.
3. Click the property value.

   The Web browser is launched using the Web address that is specified by the property value.

Part 2. Metadata reference

Chapter 5. Metadata templates

This chapter provides detailed information about each template that is provided with the Data Warehouse Center and the Information Catalog Manager. The section for each template lists the tokens for the template. It provides the allowed values and lengths of values for each token.

If your interchange program does not have a value for a token, it should set the token to `ISV_DEFAULTVALUE`. However, you must specify a value other than `ISV_DEFAULTVALUE` for any token that is required.

Because there is no template for security groups, your program must specify the value `ISV_DEFAULTSECURITYGROUP` for any instances of the **SecurityGroup* token.

If the template does not set a Data Warehouse Center parameter, the Data Warehouse Center definition will have the default value of the parameter. For example, the Data Warehouse Center sets the Retry Count and Retry Interval parameters for source databases to their default values.

Table 13 lists the metadata templates that are supplied with the Data Warehouse Center and the section that covers each template.

Table 13. Metadata templates supplied with the Data Warehouse Center

Template	Description	See:
AgentSite.tag	Defines an agent site from which the agent accesses the data source or target warehouse, or on which a Data Warehouse Center program runs.	"AgentSite.tag" on page 55
Column.tag	Defines a column or field in a table, segment, or file.	"Column.tag" on page 57
HeaderInfo.tag	Declares all the object type definitions needed by the Data Warehouse Center to declare a tag language file.	"HeaderInfo.tag" on page 63
Process.tag	Defines a process.	"Process.tag" on page 63
StarSchema.tag	Defines a star schema.	"StarSchema.tag" on page 65

Table 13. Metadata templates supplied with the Data Warehouse Center (continued)

Template	Description	See:
StarSchemaInputTable.tag	Defines the relationship between tables and a star schema.	“StarSchemaInputTable.tag” on page 67
Step.tag	Defines a step.	“Step.tag” on page 68
StepCascade.tag	Defines a cascade relationship between steps.	“StepCascade.tag” on page 72
StepInputTable.tag	Defines the relationship between a step and its source tables.	“StepInputTable.tag” on page 73
StepOutputTable.tag	Defines the relationship between a step and its target.	“StepOutputTable.tag” on page 74
StepVWPOutputTable.tag	Defines the relationship between a step and a warehouse target.	“StepVWPOutputTable.tag” on page 76
StepVWPProgramInstance.tag	Defines an instance of a specific template used by a step.	“StepVWPProgramInstance.tag” on page 77
SourceDataBase.tag	Defines a warehouse source.	“SourceDataBase.tag” on page 78
SubjectArea.tag	Defines a subject area to contain the processes and steps being created.	“SubjectArea.tag” on page 82
Table.tag	Defines a table or file that the Data Warehouse Center is to access.	“Table.tag” on page 83
VWPGroup.tag	Defines a group that is to contain any Data Warehouse Center program being defined.	“VWPGroup.tag” on page 89
VWPProgramInstanceParameter.tag	Adds or modifies a parameter that the Data Warehouse Center passes to an instance of a Data Warehouse Center program used by a specific step.	“VWPProgramInstanceParameter.tag” on page 90
VWPProgramTemplate.tag	Defines a Data Warehouse Center program.	“VWPProgramTemplate.tag” on page 93

Table 13. Metadata templates supplied with the Data Warehouse Center (continued)

Template	Description	See:
VWPProgramTemplateParameter.tag	Defines a parameter that the Data Warehouse Center is to pass to a Data Warehouse Center program.	"VWPProgramTemplateParameter.tag" on page 95
WarehouseDataBase.tag	Defines a warehouse target.	"WarehouseDataBase.tag" on page 98

AgentSite.tag

Use this template to define an agent site:

- From which the agent accesses the data sources or target warehouses.
- On which a Data Warehouse Center program runs.

You can use one of the following agent site types:

- An agent site that is already defined in the warehouse control database.

  To use an existing agent site, replace all occurrences of the **AgentSite* token with the agent site name.
- The default agent site.

  To use the default agent site, replace all occurrences of the **AgentSite* token with `ISV_DEFAULTAGENTSITE`.
- A new agent site that you define using the AgentSite.tag template.

  To define a new agent site, specify values for the tokens in the AgentSite.tag template. Replace all occurrences of the **AgentSite* token with the name of the new agent site.

Tokens

Table 14 provides information about each token in the template.

Table 14. AgentSite.tag tokens

Token	Description	Allowed values
	Entity parameters	

Table 14. AgentSite.tag tokens (continued)

Token	Description	Allowed values
**AgentSite*	The name of a new agent site, or the name of the default agent site, if the agent is not new. If you specify a new name, it must be unique within the warehouse control database. This token is required, but you can specify the default agent site, `ISV_DEFAULTAGENTSITE`	A text string, up to 80 bytes in length. If you do not want to create a new agent site, use `ISV_DEFAULTAGENTSITE` for the default agent site.
**AgentSiteContact*	The name of the person or organization responsible for this agent.	A text string.
**AgentSiteDescription*	The short description of the agent site. This token is optional.	A text string, up to 254 bytes in length.
**AgentSiteNotes*	The long description of the agent site. This token is optional.	A text string, up to 32700 bytes in length.
**AgentSiteOSType*	The type of operating system that runs on the agent site. This token is required.	One of the following values: **ISV_windowsNT** Windows NT® **ISV_AIX** AIX® **ISV_os2** OS/2® **ISV_as400** AS/400® **ISV_Solaris** SUN **ISV_MVS** MVS
**AgentSiteTCP/IPHostname*	The TCP/IP host name of the agent site. This token is required.	A text string, up to 200 bytes in length.

Table 14. AgentSite.tag tokens (continued)

Token	Description	Allowed values
**AgentSiteUserid*	The user ID under which the agent runs. This token is required.	A text string, up to 36 bytes in length.
	Relationship parameters	
**CurrentCheckPointID++*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 15 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 15. Example values for AgentSite.tag tokens

Token	Example value
**AgentSite*	My agent site
**AgentSiteContact*	DEPT W24A
**AgentSiteDescription*	This is the description of my agent site
**AgentSiteNotes*	These are the notes for my agent site.
**AgentSiteOSType*	ISV_Solaris
**AgentSiteTCP/IPHostname*	CHI11W71.stl.ibm.com
**AgentSiteUserid*	VWADMIN
**CurrentCheckPointID++*	1

Column.tag

Use this template to define a column in a table, or a field in a segment or file. You can use this template to define columns or fields for both sources and targets.

The template defines the relationship between the column or field and the table, segment, or file that contains the column or field. You must include this template if you defined sources or targets by using the Table.tag template (see page 83).

Tokens

Table 16 on page 58 provides information about each token in the template.

Column.tag

Table 16. Column.tag tokens

Token	Description	Allowed values
	Entity parameters	
**ColumnName*	The name of the column or field. The name must be unique within a table or field. This token is required.	A text string, up to 80 bytes in length.
**ColumnDescription*	The short description of the column or field. This token is optional.	A text string, up to 254 bytes in length.
**ColumnNotes*	The long description of the column or field. This token is optional.	A text string, up to 32700 bytes in length.
**ColumnOffsetFromZero*	The offset in bytes from the start of the file to where the data for this field starts.	A numeric value or 0.
**ColumnOrdinalNumber*	The ordinal position of the column. Usually the same as the **ColumnPositionNumber*.	A numeric value or 0.
**ColumnUserActions*	The actions that a user can perform on this column or field. This token is optional.	A text string, up to 254 bytes in length.
**ColumnLength*	The length of the column or field being created. This token is required.	A numeric value.
**ColumnPrecision*	The precision of the column or field for columns or fields with a decimal data type. This token is required.	A numeric value or 0.
**ColumnKeyPosition*	If this column is part of a key, the column's position within the key. This token is required.	A numeric value. If there is no precision value, specify 0.

Table 16. Column.tag tokens (continued)

Token	Description	Allowed values
**ColumnPositionNumber*	A number, starting with 0, that indicates the order of the column within the row. This token is required.	A numeric value.
**ColumnAllowsNulls*	A flag that specifies whether the column or field allows null data. This token is required.	One of the following values: **ISV_NULLSYES** The column allows null data. **ISV_NULLSNO** The column does not allow null data.
**ColumnDataIsText*	A flag that specifies whether the column or field contains only text data for character types. This token is required.	One of the following values: **ISV_ISTEXTYES** The column contains only text data. **ISV_ISTEXTNO** The column does not contain only text data.
**ColumnEditionType*	Identifies whether the column holds Data Warehouse Center edition information.	One of the following values: **ISV_ColumnIsEditionColumn** The column is an edition column. **ISV_ColumnIsNormal** The column is a normal column.

Table 16. Column.tag tokens (continued)

Token	Description	Allowed values
**ColumnNativeDataType*	The data type of the column or field as defined to the database manager or file system. This token is required.	One of the following values: ISV_NATIVE_CHAR ISV_NATIVE_VARCHAR ISV_NATIVE_LONGVARCHAR ISV_NATIVE_VARCHAR2 ISV_NATIVE_GRAPHIC ISV_NATIVE_VARGRAPHIC ISV_NATIVE_LONGVARGRAPHIC ISV_NATIVE_CLOB ISV_NATIVE_INT ISV_NATIVE_TINYINT ISV_NATIVE_BLOB ISV_NATIVE_SMALLINT ISV_NATIVE_INTEGER ISV_NATIVE_FLOAT ISV_NATIVE_SMALLFLOAT ISV_NATIVE_DOUBLE ISV_NATIVE_REAL ISV_NATIVE_DECIMAL ISV_NATIVE_SMALLMONEY ISV_NATIVE_MONEY ISV_NATIVE_NUMBER

Table 16. Column.tag tokens (continued)

Token	Description	Allowed values
**ColumnNativeDataType* (continued)	The data type of the column or field as defined to the database manager or file system. This token is required.	One of the following values: ISV_NATIVE_NUMERIC ISV_NATIVE_DATE ISV_NATIVE_TIME ISV_NATIVE_TIMESTAMP ISV_NATIVE_LONG ISV_NATIVE_RAW ISV_NATIVE_LONGRAW ISV_NATIVE_DATETIME ISV_NATIVE_SMALLDATETIME ISV_NATIVE_SYSNAME ISV_NATIVE_TEXT ISV_NATIVE_BINARY ISV_NATIVE_VARBINARY ISV_NATIVE_LONGVARBINARY ISV_NATIVE_BIT ISV_NATIVE_IMAGE ISV_NATIVE_SERIAL ISV_NATIVE_DBCLOB ISV_NATIVE_BIGINT ISV_NATIVE_DATETIMEYEARTOFRACTION
Relationship parameters		
**CurrentCheckPointID++*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.
**DatabaseName*	The business name of the warehouse source or warehouse target. This token is required.	A text string, up to 40 bytes in length.
**TablePhysicalName*	The physical name of the table or file that contains the column as defined to the database manager or file system. This token is required.	A text string, up to 80 bytes in length.

Table 16. Column.tag tokens (continued)

Token	Description	Allowed values
**TableOwner*	The owner, high-level qualifier, collection, or schema of the table that contains the column. This token is required.	A text string, up to 15 bytes in length.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 17 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 17. Example values for Column.tag tokens

Token	Example value
**ColumnName*	Country_code
**ColumnDescription*	This column contains the country code
**ColumnNotes*	The valid values for this column can be found in the Geography reference manual
**ColumnOffsetFromZero*	0
**ColumnOrdinalNumber*	0
**ColumnUserActions*	User cannot directly view a single column
**ColumnLength*	10
**ColumnPrecision*	0
**ColumnKeyPosition*	0
**ColumnAllowsNulls*	ISV_NULLSNO
**ColumnDataIsText*	ISV_ISTEXTYES
**ColumnNativeDataType*	ISV_NATIVE_CHAR
**DatabasePhysicalName*	FINANCE
**TableOwner*	DB2ADMIN
**TablePhysicalName*	GEOGRAPHY
**CurrentCheckPointID++*	8

HeaderInfo.tag

Use this template to declare all of the object type definitions that are needed by the Data Warehouse Center to process a tag language file. The template also contains definitions that the Data Warehouse Center associates with other definitions, such as the security group that is to contain the objects that you are importing. This template is always required and must be present in the beginning of the tag language file.

Tokens

Table 18 provides information about each token in the template.

Table 18. HeaderInfo.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values
**SecurityGroup*	The security group that is to contain all the objects that you are importing. This token is required, and you must specify the default security group.	`ISV_DEFAULTSECURITYGROUP` for the default security group.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 19 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 19. Example values for Header.tag tokens

Token	Example value
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**CurrentCheckPointID++*	0

Process.tag

Use this template to define a process to group steps. Each step must be in only one process. The process is related to subject areas, and each partner application must have at least one subject area that any processes resides in. The template defines the relationship between the subject area and the partner application's security group as well as between the process and the subject area.

Process.tag

This template is required if the partner application is defining steps to the Data Warehouse Center.

If you create a new process object, the value that you provide for the **ProcessName* token must be unique to all processes defined in the warehouse control database.

Tokens

Table 20 provides information about each token in the template.

Table 20. Process.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values
	Entity parameters	
**ProcessName*	The unique name of the process.	A text string, up to 80 bytes in length.
**ProcessDescription*	The description that is associated with the process.	A text string, up to 254 bytes in length.
**ProcessNotes*	The long description that is associated with the process.	A text string, up to 32,700 bytes in length.
**ProcessContact*	The name of a person or group to contact for questions or concerns about this step.	A text string.
**ProcessType*	The processing options if there was no source data.	One of the following values: **ISV_ProcessType_Normal** Process is a normal user process.
	Relationship parameters	
**SubjectArea*	The name of a subject area that is to contain this process and thesteps being created or being added to this process.	A text string, up to 80 bytes in length.
**SecurityGroup*	The security group that is to contain all the objects that you are importing. This token is required, and you must specify the default security group.	`ISV_DEFAULTSECURITYGROUP` for the default security group.

Table 20. Process.tag tokens (continued). This template contains only relationship parameters.

Token	Description	Allowed values
**CurrentCheckPointID*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 21 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 21. Example values for Process.tag tokens

Token	Example value
**ProcessName*	Marketing process
**ProcessDescription*	A collection of steps that is used by the marketing organization
**ProcessNotes*	Steps that create the star schema that is used by the marketing organization
**ProcessContact*	Marketing
**ProcessType*	ISV_ProcessType_2
**SubjectArea*	Group of processes generated for this partner application
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**CurrentCheckPointID*	9

StarSchema.tag

Use this template to define a star schema as a mechanism to group tables that are related. The star schema can be used to relate tables within the same physical database (for further use by the DB2 OLAP Integration Server) or for logical grouping by relating tables from multiple databases.

Tokens

Table 22 provides information about each token in the template.

Table 22. StarSchema.tag tokens

Token	Description	Allowed values
	Entity parameters	

Table 22. StarSchema.tag tokens (continued)

Token	Description	Allowed values
**StarSchemaName*	The unique name of the star schema that is being created or related.	A text string, up to 80 bytes in length.
**StarSchemaDescription*	A description that is associated with the star schema.	A text string, up to 254 bytes in length.
**StarSchemaNotes*	The long description that is associated with the step.	A text string, up to 32,700 bytes in length.
**StarSchemaContact*	The name of a person or group to contact for questions or concerns about this step.	A text string.
**StarSchemaDBName*	The business name of the database that is being created.	A text string.
	Relationship parameters	
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 23 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 23. Example values for StarSchema.tag tokens

Token	Example value
**StarSchemaName*	Marketing schema
**StarSchemaDescription*	This star schema represents the marketing division's internal databases
**StarSchemaNotes*	Tables used for the marketing division
**StarSchemaContact*	Marketing group
**StarSchemaDBName*	Marketing
**CurrentCheckPointID++*	3

StarSchemaInputTable.tag

Use this template to define the relationship between a star schema and its input source. This relationship is required for all star schemas.

Tokens

Table 24 provides information about each token in the template.

Table 24. SourceDataBase.tag tokens

Token	Description	Allowed values
	Entity parameters	
**StarSchemaName*	The name of the star schema that is being created or related.	A text string.
	Relationship parameters	
**DatabaseName*	The business name of the database that is being created.	A text string.
**TableOwner*	The owner, high-level qualifier, collection, or schema of the table that is being described. This value must be a valid qualifier as defined by the rules of ODBC.	A text string.
**TablePhysicalName*	The physical table name as it is known to ODBC (the system DSN name).	A text string.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 25 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 25. Example values for SourceDataBase.tag tokens

Token	Example value
**StarSchemaName*	Finance schema
**DatabaseName*	Finance Warehouse
**TableOwner*	DB2ADMIN

Table 25. Example values for SourceDataBase.tag tokens (continued)

Token	Example value
**TablePhysicalName*	DB2ADMIN.GEOGRAPHY
**CurrentCheckPointID++*	7

Step.tag

Use this template to define a step that will be managed by the Data Warehouse Center. This template includes information about the relationships to security group, process, and agent site.

This template is required for all partner applications that are generating relationships between source and target data or defining programs that the Data Warehouse Center is to run.

If you create a new step object, the value that you provide for the **StepName* token must be unique to all steps that are defined in the warehouse control database.

Tokens

Table 26 provides information about each token in the template.

Table 26. Step.tag tokens

Token	Description	Allowed values
	Entity parameters	
**StepName*	The unique name of the step that is being created or related.	A text string, up to 80 bytes in length.
**StepDescription*	The description that is associated with the step.	A text string, up to 254 bytes in length.
**StepNotes*	The long description that is associated with the step.	A text string, up to 32,700 bytes in length.

Table 26. Step.tag tokens (continued)

Token	Description	Allowed values
**StepDataNotPresent*	The processing options if there was no source data.	One of the following values: **ISV_StepDataNotPresent_OK** If data is not present, continue processing. **ISV_StepDataNotPresent_Warning** If data is not present, issue a warning and continue processing. **ISV_StepDataNotPresent_Error** If data is not present, issue an error message and stop processing.
**StepSelectStatementGenerated*	A flag that specifies whether the Data Warehouse Center is to generate the SQL, or whether the SQL statement is included by the token **StepSelectStatement*.	One of the following values: **ISV_StepSelectStatementNo** The SELECT statement is not generated, but is included in the **StepSelectStatement.* **ISV_StepSelectStatementYes** The SELECT statement is generated, and **StepSelectStatement* is ignored.
**StepSelectStatement*	The SQL statement to be issued if ISV_StepSelectStatementNo.	A SQL string.
**StepContact*	The name of a person or group to contact for questions or concerns about this step.	A text string.
**StepExternalPopulation*	A flag that indicates that the step is expected to be run outside the Data Warehouse Center environment..	One of the following values: **ISV_StepExternalNo** The table will not be externally populated by other means. **ISV_StepExternalYes** The table will be externally populated by other means.

Table 26. Step.tag tokens (continued)

Token	Description	Allowed values
**StepType*	The type of step that is being created.	One of the following values: **ISV_StepType_Editioned_Append** The data in the table will be appended when the Step is run. **ISV_StepType_Full_Replace** The data in the table will be replaced when the Step is run. **ISV_StepType_Uneditioned_Append** The data in the table will be appended when the Step is run. **ISV_StepType_VWP_Population** The data in the table is populated by a Data Warehouse Center program.
**StepSQLWarning*	The processing options if an SQL warning occurs.	One of the following values: **ISV_StepSQLWarning_OK** If an SQL warning occurs, continue processing. **ISV_StepSQLWarning_Warning** If an SQL warning occurs, issue a warning and continue processing. **ISV_StepSQLWarning_Error** If an SQL warning occurs, issue an error and stop processing.
**StepCommit*	A flag that specifies if the Data Warehouse Center is to intermittently commit after **StepCommitAfterNumberRows* is inserted into the target table of the step.	One of the following values: **ISV_Step_Incremental_Commit_On** The data is to be incrementally committed at the target. **ISV_Step_Incremental_Commit_Off** The data is not to be incrementally committed at the target.

Table 26. Step.tag tokens (continued)

Token	Description	Allowed values
**StepCommitAfterNumberRows*	The number of rows to insert before committing.	A numeric value.
	Relationship parameters	
**SecurityGroup*	The security group that is to contain all the objects that you are importing. This token is required, and you must specify the default security group.	`ISV_DEFAULTSECURITYGROUP` for the default security group.
**ProcessName*	The name of the process. This token is required.	A text string, up to 80 bytes in length.
**AgentSite*	The name of a new agent site, or the name of the default agent site, if the agent is not new. If you specify a new name, it must be unique within the Data Warehouse Center control database. This token is required, but you can specify the default agent site, `ISV_DEFAULTAGENTSITE`	A text string, up to 80 bytes in length. If you do not want to create a new agent site, use `ISV_DEFAULTAGENTSITE` for the default agent site.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 27 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 27. Example values for Step.tag tokens

Token	Example value
**StepName*	Revenue by location
**StepDescription*	This step will pull data to create the revenue for each location in a DB2 table
**StepNotes*	Revenue for Geography 7 comes from 4 source Oracle tables

Table 27. Example values for Step.tag tokens (continued)

Token	Example value
**StepDataNotPresent*	ISV_StepDataNotPresent_Error
**StepSelectStatementGenerated*	ISV_StepSelectStatementNo
**StepSelectStatement*	SELECT * FROM IWH.REVENUE_BY_LOCATION
**StepContact*	Jason Smythe
**StepExternalPopulation*	ISV_StepExternalNo
**StepType*	ISV_StepType_Full_Replace
**StepSQLWarning*	ISV_StepSQLWarning_Warning
**StepCommit*	ISV_Step_Incremental_Commit_On
**StepCommitAfterNumberRows*	10000
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**ProcessName*	Marketing process
**AgentSite*	My agent site
**CurrentCheckPointID++*	5

StepCascade.tag

Use this template to define a relationship between two steps to specify that another step is to be started at the completion of the named step.

This template is required only if the partner application links steps in a cascaded relationship.

Tokens

Table 28 provides information about each token in the template.

Table 28. StepCascade.tag tokens

Token	Description	Allowed values
	Entity parameters	
**StepName*	The name of the step that is being related.	A text string.
**PostStepName*	The name of the step that is to be run after the completion of another step.	A text string.
	Relationship parameters	

Table 28. StepCascade.tag tokens (continued)

Token	Description	Allowed values
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 29 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 29. Example values for StepCascade.tag tokens

Token	Example value
**StepName*	Revenue by location
**PostStepName*	Revenue for all Geographies
**CurrentCheckPointID++*	12

StepInputTable.tag

This template defines the relationship between a step and its input source.

This relationship is required for steps of type ISV_StepType_Editioned_Append, ISV_StepType_Full_Replace, and ISV_StepType_Uneditioned_Append.

This relationship is optional for steps of type ISV_StepType_VWP_Population.

The ISV can relate multiple input sources to the step by reusing the template for each unique instance of an input source.

Tokens

Table 30 provides information about each token in the template.

Table 30. StepInputTable.tag tokens

Token	Description	Allowed values
	Entity parameters	
**StepName*	The name of the step that is being related.	A text string.
	Relationship parameters	
**DatabaseName*	The business name of the database that is being created.	A text string.

Table 30. StepInputTable.tag tokens (continued)

Token	Description	Allowed values
**TableOwner*	The owner, high-level qualifier, collection, or schema of the table that is being described. This value must be a valid qualifier as defined by the rules of ODBC.	A text string.
**TablePhysicalName*	The physical table name as it is known to ODBC (the system DSN name).	A text string.
**ProcessName*	The name of the process that is being related.	A text string.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 31 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 31. Example values for StepInputTable.tag tokens

Token	Example value
**StepName*	Revenue by product
**DatabaseName*	Finance Warehouse
**TableOwner*	FINADMIN
**TablePhysicalName*	INVENTORY
**ProcessName*	Inventory process
**CurrentCheckPointID++*	6

StepOutputTable.tag

Use this template to define the relationship between a step and its output target.

This relationship is required for steps of type ISV_StepType_Editioned_Append, ISV_StepType_Full_Replace, ISV_StepType_Uneditioned_Append.

Tokens

Table 32 provides information about each token in the template.

Table 32. SourceDataBase.tag tokens

Token	Description	Allowed values
	Entity parameters	
**StepName*	The name of the step that is being created or related.	A text string.
	Relationship parameters	
**DatabaseName*	The business name of the database that is being related.	A text string.
**TableOwner*	The owner, high-level qualifier, collection, or schema of the table being described. This value must be a valid qualifier as defined by the rules of ODBC.	A text string.
**TablePhysicalName*	The physical table name as it is known to ODBC (the system DSN name).	A text string.
**ProcessName*	The name of the process that is being related.	A text string.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 33 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 33. Example values for StepOutputTable.tag tokens

Token	Example value
**StepName*	Revenue by product
**DatabaseName*	Finance Warehouse
**TableOwner*	FINADMIN
**TablePhysicalName*	INVENTORY
**ProcessName*	Marketing process

Table 33. Example values for StepOutputTable.tag tokens (continued)

Token	Example value
**CurrentCheckPointID++*	4

StepVWPOutputTable.tag

Use this template to optionally define the relationship between a step of type ISV_StepType_VWP_Population and its output targets.

Tokens

Table 34 provides information about each token in the template.

Table 34. StepVWPOutputTable.tag tokens

Token	Description	Allowed values
	Entity parameters	
**StepName*	The name of the step that is being related.	A text string.
	Relationship parameters	
**DatabaseName*	The business name of the database that is being created.	A text string.
**TableOwner*	The owner, high-level qualifier, collection, or schema of the table that is being described. This value must be a valid qualifier as defined by the rules of ODBC.	A text string.
**TablePhysicalName*	The physical table name as it is known to ODBC (the system DSN name).	A text string.
**ProcessName*	The name of the processthat is being created or related	A text string.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 35 on page 77 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 35. Example values for StepVWPOutputTable.tag tokens

Token	Example value
**StepName*	Revenue by product
**DatabaseName*	Finance Warehouse
**TableOwner*	FINADMIN
**TablePhysicalName*	INVENTORY
**ProcessName*	Marketing process
**CurrentCheckPointID++*	1

StepVWPProgramInstance.tag

Use this template to define an instance of a Data Warehouse Center program that is run by a warehouse agent. This template also defines the relationship to the Data Warehouse Center program definition, called the VWPTemplate, as well as the step that uses the Data Warehouse Center program. This template is required for each step that utilizes the Data Warehouse Center program.

Tokens

Table 36 provides information about each token in the template.

Table 36. StepVWPProgramInstance.tag tokens

Token	Description	Allowed values
	Entity parameters	
**VWPProgramInstanceKey*	Key that uniquely identifies this program instance. The key must be unique from all other keys in the tag language file. **Tip**: Finish processing the VWPProgramInstance.tag template before increasing the value of the key. This token is required.	A numeric value.
	Relationship parameters	
**StepName*	The name of the step that is being related.	A text string.
**VWPProgramTemplateName*	The business name of the Data Warehouse Center program template that is being created.	A text string.

Table 36. StepVWPProgramInstance.tag tokens (continued)

Token	Description	Allowed values
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 37 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 37. Example values for StepVWPProgramInstance.tag tokens

Token	Example value
**VWPProgramInstanceKey*	070001
**StepName*	Revenue by location
**VWPProgramTemplateName*	My ISV Program
**CurrentCheckPointID++*	2

SourceDataBase.tag

Use this template to define source databases, file systems, or files to import into the Data Warehouse Center. You can use this template to define a relational non-DB2 source database as well as a DB2 source database.

This template also defines the relationship between the following objects:

- The source databases
- The agent site to use for the source database
- The security group in which to define the source database

Tokens

Table 38 provides information about each token in the template.

Table 38. SourceDataBase.tag tokens

Token	Description	Allowed values
	Entity parameters	

Table 38. SourceDataBase.tag tokens (continued)

Token	Description	Allowed values
**DatabaseName*	The name of the database. The name must be unique within the warehouse control database. This token is required.	A text string, up to 80 bytes in length.
**DatabaseDescription*	The short description of the database. This token is optional.	A text string, up to 254 bytes in length.
**DatabaseNotes*	The long description of the database. This token is optional.	A text string, up to 32700 bytes in length.
**DatabaseContact*	The person to contact for information about this database. This token is optional.	A text string, up to 64 bytes in length.
**DatabaseServerName*	The name of the server on which the database resides. This token is required for Flat File LAN files. Otherwise, it is optional.	A text string, up to 64 bytes in length.
**DatabaseVersion*	The version of the database.	A text string.
**DatabasePhysicalName*	The physical database name of the database as defined to the database manager, as known to ODBC. This token is required.	A text string, up to 40 bytes in length.

Table 38. SourceDataBase.tag tokens (continued)

Token	Description	Allowed values
**DatabaseType*	The type of database family. This token is required.	One of the following values: **ISV_IR_DB2Family** DB2 Family **ISV_IR_Oracle** Oracle **ISV_IR_Sybase** Sybase **ISV_IR_MSSQLServer** Microsoft® SQLServer **ISV_IR_Informix** Informix **ISV_IR_GenericODBC** Generic ODBC **ISV_IR_FFLan** Flat File LAN **ISV_IR_VSAM** VSAM **ISV_IR_IMS** IMS
**DatabaseTypeExtended*	The type of AS/400 system or file. This token is required.	One of the following values: **ISV_IR_DB2400CISC** DB2 UDB for AS/400 for CISC **ISV_IR_DB2400RISC** DB2 UDB for AS/400 for RISC **ISV_IR_FFLanLocalCmd** Local flat file **ISV_IR_FFLanFTPCopy** Local flat file sent using FTP from a remote system
**DatabaseUserid*	The user ID with which to access the database. This token is optional.	A text string, up to 36 bytes in length.
	Relationship parameters	

Table 38. SourceDataBase.tag tokens (continued)

Token	Description	Allowed values
**SecurityGroup*	The security group in which to create the source or target database. This token is required, and you must specify the default security group.	`ISV_DEFAULTSECURITYGROUP` for the default security group.
**AgentSite*	The agent site to use for the source or target database. This token is required, but you can specify the default agent site.	A text string, up to 80 bytes in length. `ISV_DEFAULTAGENTSITE` for the default agent site.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 39 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 39. Example values for SourceDataBase.tag tokens

Token	Example value
**DatabaseName*	Finance Warehouse
**DatabaseDescription*	This database contains financial information.
**DatabaseNotes*	This is the warehouse where all geographies keep financial information.
**DatabaseContact*	Valerie Zieman
**DatabaseServerName*	CHI11W71
**DatabaseVersion*	V6.1.0
**DatabasePhysicalName*	FINANCE
**DatabaseType*	ISV_IR_DB2Family
**DatabaseTypeExtended*	ISV_DEFAULTVALUE
**DatabaseUserid*	DB2ADMIN
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**AgentSite*	My agent site

Table 39. Example values for SourceDataBase.tag tokens (continued)

Token	Example value
**CurrentCheckPointID++*	5

SubjectArea.tag

Use this template to define a subject area to contain the processes and steps that you create. Each tag language file must have at least one subject area to contain any processes and steps that you create. This template is required if you are defining processes and steps.

This template also defines the relationship between the subject area and the security group that the header file specifies (see "HeaderInfo.tag" on page 63).

Tokens

Table 40 provides information about each token in the template.

Table 40. SubjectArea.tag tokens

Token	Description	Allowed values
	Entity parameters	
**SubjectArea*	The name of a group that is to contain all of the processes and steps that are created or added to a particular subject area. The name must be unique within the warehouse control database. This token is required.	A text string, up to 80 bytes in length.
**SubjectAreaContact*	The name of the person or organization that is responsible for this subject area.	A text string.
**SubjectAreaDescription*	A short description of the group of processes and steps. This token is optional.	A text string, up to 254 bytes in length.
**SubjectAreaNotes*	A long description of the group of processes and steps. This token is optional.	A text string, up to 32700 bytes in length.

Table 40. SubjectArea.tag tokens (continued)

Token	Description	Allowed values
	Relationship parameters	
**SecurityGroup*	The security group in which to create the subject area. This token is required, and you must specify the default security group.	`ISV_DEFAULTSECURITYGROUP` for the default security group.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 41 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 41. Example values for SubjectArea.tag tokens

Token	Example value
**SubjectArea*	Group of processes and steps generated for the partner tool
**SubjectAreaContact*	DEPT W24A
**SubjectAreaDescription*	This subject area contains all the processes and steps generated for Data Warehouse Center by the partner tool.
**SubjectAreaNotes*	The processes and steps in this subject area will be used to evaluate the product.
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**CurrentCheckPointID++*	9

Table.tag

You can use this template to define both source and target tables as well as source files and segments that Data Warehouse Center is to access. You can use this template to define source and target tables, files, and segments.

The template defines all the metadata that the Data Warehouse Center requires to define a table in an ODBC data source as well as a DB2 target table. The template also defines the relationships between the table and the database that contains the table.

Tokens

Table 42 provides information about each token in the template.

Table 42. Table.tag tokens

Token	Description	Allowed values
	Entity parameters	
**TableFullName*	The fully qualified name of a relational table or a file. For a table, this name is the concatenation of the value of the **TableOwner* and **TablePhysicalName* tokens, separated by a period. For a file, the **TableOwner* value should be left blank, and the **TableFullName* and **TablePhysicalName* values should be the same. The name must be unique within the warehouse control database. This token is required.	A text string, up to 80 bytes in length.
**TableDescription*	The short description of the table. This token is optional.	A text string, up to 254 bytes in length.
**TableNotes*	The long description of the table. This token is optional.	A text string, up to 32700 bytes in length.
**TableOwner*	The owner, high-level qualifier, collection, or schema of the table. This token is required.	A text string, up to 15 bytes in length.
**TablePhysicalName*	The physical table name as defined to the database manager or file system. This token is required.	A text string, up to 80 bytes in length.

Table 42. Table.tag tokens (continued)

Token	Description	Allowed values
**TableBinaryIfFile*	A flag that specifies whether the file contains only binary data if the table represents a file. This token is optional.	One of the following values: **ISV_DR_FILE_IS_BINARY** The file is binary. **ISV_DR_FILE_IS_NOT_BINARY** The file is in ASCII or mixed format.
**TableFirstRowNamesIfFile*	A flag that specifies whether the first row of the file contains column names if the table represents a file. This token is optional.	One of the following values: **ISV_DR_ROW_CONTAINS_NAMES** The first row of the file contains column names. **ISV_DR_ROW_DOES_NOT_CONTAIN_NAMES** The first row of the file contains data.
**TableTypeIfFile*	The type of file if the table represents a file. This token is optional.	One of the following values: **ISV_DR_REL_TABLE** The table is a relational table. **ISV_DR_COMMA_DELIMITED** The columns in the file are separated by commas. **ISV_DR_FIXED_FORMAT** The columns in the file are in fixed format. **ISV_DR_TAB_DELIMITED** The columns in the file are separated by tabs. **ISV_DR_CHAR_DELIMITED** The columns in the file are separated by the value of **TableDelimiterIfFile*.
**TableDelimiterIfFile*	The value of the delimiter to separate fields if the file type is ISV_DR_CHAR_DELIMITED. This token is optional.	A text string, 1 byte in length.

Table 42. Table.tag tokens (continued)

Token	Description	Allowed values
**TableIsAView*	A token that specifies whether the table is a view.	One of the following values: **ISV_TableIsAView** The table is a view. **ISV_TableIsNotAView** The table is not a view.
**TableIsADimensionTable*	A token that specifies whether the table is a part of a star schema and contains dimensional data.	One of the following values: **ISV_TableIsADimensionalTable** The table is a dimensional table. **ISV_TableIsNotADimensionalTable** The table is not a dimensional table.
**TableIsAnAlias*	A token that specifies whether the table is actually an alias of another table.	One of the following values: **ISV_TableIsAnAlias** This table is an alias for another table. **ISV_TableIsNotAnAlias** This table is not an alias for another table.
**TableCreatedByDWC*	A token that specifies whether the Data Warehouse Center should create and manage this table.	One of the following values: **ISV_TableIsToBeCreatedByDWC** The table is to be created by the Data Warehouse Center. **ISV_TableIsNotToBeCreatedByDWC** The table is not to be created by the Data Warehouse Center.
**TableGrantedToPublic*	A token that specifies whether the Data Warehouse Center should grant public access to this table when the table is created. This is only valid if the Data Warehouse Center creates the table.	One of the following values: **ISV_GrantTableAccessToPublic** The Data Warehouse Center is to grant PUBLIC access to this table. **ISV_DoNotGrantTableAccessToPublic** The Data Warehouse Center is not to grant PUBLIC access to this table.

Table 42. Table.tag tokens (continued)

Token	Description	Allowed values
**TableIsPersistent*	A token that specifies whether the data in the table is to persist between executions of the steps that use this table. If the table is not persistent, the data in the table will be deleted after each use.	One of the following values: **ISV_TableIsPersistent** The table is to be considered persistent. **ISV_TableIsTransient** The table is to be considered transient.
**TableMaximumEditions*	The maximum number of editions the table is to have, if the table supports editions.	A numeric value.
**TableGenerateCreateStatement*	A token that specifies whether the Data Warehouse Center is to generate the create table statement.	One of the following values: **ISV_GenerateCreateTableStmt** The Data Warehouse Center should generate the CREATE TABLE statement. **ISV_DoNotGenerateCreateTableStmt** The Data Warehouse Center should not generate the CREATE TABLE statement.
**TableIsAFactTable*	A token that specifies whether the table is part of a star schema, and the table contains the fact information.	One of the following values: **ISV_TableIsAFactTable** The table is a fact table. **ISV_TableIsNotAFactTable** The table is not a fact table.
**TableCreateStatement*	The DDL to create the table. Use this token only if the ISV_DoNotGenerateCreateTableStmt has been specified.	A text string.
	Relationship parameters	

Table 42. Table.tag tokens (continued)

Token	Description	Allowed values
**DatabaseName*	The name of the database that contains the table. The name must be unique within the warehouse control database. This token is required.	A text string, up to 80 bytes in length.
**DatabasePhysicalName*	The physical database name of the database that contains the table. This token is required.	A text string, up to 40 bytes in length.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 43 provides examples of values for each token to illustrate the kind of metadata that you might provide for each token.

Table 43. Example values for Table.tag tokens

Token	Example value
**TableFullName*	DB2ADMIN.GEOGRAPHY
**TableDescription*	Contains geography information
**TableNotes*	This table contains all the information about geographies serviced by our company
**TableOwner*	DB2ADMIN
**TablePhysicalName*	GEOGRAPHY
**TableBinaryIfFile*	ISV_DEFAULTVALUE
**TableFirstRowNamesIfFile*	ISV_DEFAULTVALUE
**TableTypeIfFile*	ISV_DEFAULTVALUE
**TableDelimiterIfFile*	ISV_DEFAULTVALUE
**TableIsAView*	ISV_TableIsAView
**TableIsADimensionTable*	ISV_TableIsNotADimensionTable
**TableIsAnAlias*	ISV_TableIsAnAlias

Table 43. Example values for Table.tag tokens (continued)

Token	Example value
**TableCreatedByDWC*	ISV_TableIsToBeCreatedByDWC
**TableGrantedToPublic*	ISV_GrantTableAccessToPublic
**TableIsPersistent*	ISV_TableIsTransient
**TableMaximumEditions*	12
**TableGenerateCreateStatement*	ISV_GenerateCreateTableStmt
**TableIsAFactTable*	ISV_TableIsAFactTable
**TableCreateStatement*	Create table xyz
**DatabaseName*	Finance warehouse
**DatabasePhysicalName*	FINANCE
**CurrentCheckPointID++*	7

VWPGroup.tag

Use this template to define a group that is to contain any Data Warehouse Center programs that you are defining. This template is required if you are defining Data Warehouse Center programs.

Tokens

Table 44 provides information about each token in the template.

Table 44. VWPGroup.tag tokens

Token	Description	Allowed values
	Entity parameters	
**VWPGroup*	The unique name of a program group that is to contain all of the Data Warehouse Center programs being created. The name must be unique within the warehouse control database. This token is required.	A text string, up to 80 bytes in length.
**VWPGroupDescription*	The short description of the group of Data Warehouse Center programs. This token is optional.	A text string, up to 254 bytes in length.

Table 44. VWPGroup.tag tokens (continued)

Token	Description	Allowed values
**VWPGroupNotes*	The long description of the group of Data Warehouse Center programs. This token is optional.	A text string, up to 32700 bytes in length.
	Relationship parameters	
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 45 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 45. Example values for VWPGroup.tag tokens

Token	Example value
**VWPGroup*	Group of programs for the partner tool
**VWPGroupDescription*	This group contains all the programs used by Data Warehouse Center for the partner tool
**VWPGroupNotes*	These programs can be used to determine the relationship between sales and location.
**CurrentCheckPointID++*	2

VWPProgramInstanceParameter.tag

Use this template to add or change a parameter that the Data Warehouse Center passes to an instance of a Data Warehouse Center program for a specific step. For example, you set a default value for a host name parameter in the VWPProgramTemplateParameter.tag file (see page 95). You use this template to change the value that is passed to the Data Warehouse Center program when this particular step runs.

This template is required if the Data Warehouse Center program requires the Data Warehouse Center to pass parameters to it. You can specify that the Data Warehouse Center pass multiple parameters to the program by including this template for each parameter.

The template also defines the relationship between the parameter and its program instance.

Tokens

Table 46 provides information about each token in the template.

Table 46. VWPProgramInstanceParameter.tag tokens

Token	Description	Allowed values
	Entity parameters	
**VWPProgramInstanceParameterName*	The unique name or description of a parameter that is to be passed to a Data Warehouse Center program. This token is required.	A text string, up to 80 bytes in length.
**VWPProgramInstanceParameterOrder*	A number, starting with 0, that indicates the order of the parameter in the parameter list. This token is required.	A numeric value.
**VWPProgramInstanceParameterData*	The data that is passed to the Data Warehouse Center program as the value of the parameter. This token is required.	A text string or a numeric value up to 240 bytes in length.
**VWPProgramInstanceParameterKey*	A key that uniquely identifies this program parameter instance. The key must be unique from all other parameter keys in the interchange file. **Tip:** Finish processing the VWPProgramInstanceParameter.tag template before increasing the value of the key. This token is required.	A text value, up to 10 bytes in length.

Table 46. VWPProgramInstanceParameter.tag tokens (continued)

Token	Description	Allowed values
**VWPProgramInstanceParameterType*	The type of value that this parameter contains. For example, character, numeric, or password data.	One of the following values: **ISV_ParameterTypeNone** The parameter type is unknown or not applicable. **ISV_ParameterTypeCharacter** The parameter type is character. **ISV_ParameterTypeNumeric** The parameter type is numeric. **ISV_ParameterTypePassword** The parameter type is password.
	Relationship parameters	
**VWPProgramInstanceKey*	A key that uniquely identifies this program instance. The key must be unique from all other keys in the interchange file. **Tip:** Finish processing the VWPProgramInstance.tag template before increasing the value of the key. This token is required.	A text value, up to 10 bytes in length
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

The following table provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 47. Example values for VWPProgramInstanceParameter.tag tokens

Token	Example value
**VWPProgramInstanceParameterName*	DB2 UDB user ID
**VWPProgramInstanceKey*	070000

Table 47. Example values for VWPProgramInstanceParameter.tag tokens (continued)

Token	Example value
**VWPProgramInstanceParameterOrder++*	1
**VWPProgramInstanceParameterData*	my_userid
**VWPProgramInstanceParameterKey*	012994
**VWPProgramInstanceParameterType*	ISV_ParameterTypeNumeric
**VWPProgramInstanceKey*	070001
**CurrentCheckPointID++*	12

VWPProgramTemplate.tag

Use this template to define a Data Warehouse Center program. This template is required if the tag language file refers to a Data Warehouse Center program, unless the warehouse program already exists in the Data Warehouse Center control database.

The template also defines the relationship between the warehouse program definition and the Data Warehouse Center program group to which the program belongs.

Tokens

Table 48 provides information about each token in the template.

Table 48. VWPProgramTemplate.tag tokens

Token	Description	Allowed values
	Entity parameters	
**VWPProgramTemplateName*	The name of the Data Warehouse Center program template. The name must be unique within the warehouse control database. This token is required.	A text string, up to 80 bytes in length.
**VWPProgramTemplateDescription*	The short description of the Data Warehouse Center program and what it does. This token is optional.	A text string, up to 254 bytes in length.

Table 48. VWPProgramTemplate.tag tokens (continued)

Token	Description	Allowed values
**VWPProgramTemplateNotes*	The long description of the Data Warehouse Center program and what it does. This token is optional.	A text string, up to 32700 bytes in length.
**VWPProgramTemplateExecutableName*	The fully qualified program name of the Data Warehouse Center program that is to run when the step runs. If the Data Warehouse Center program is installed in the system path, the warehouse program name need not be fully qualified. This token is required.	A text string, up to 240 bytes in length.
**VWPProgramTemplateType*	The type of program. This token is required.	One of the following values: **ISV_PROGRAMTYPEDLL** The Data Warehouse Center program is loaded from a dynamic link library (DLL) or is a load module. **ISV_PROGRAMTYPECOMMAND** The Data Warehouse Center program is a command file. **ISV_PROGRAMTYPEEXECUTABLE** The Data Warehouse Center program is an executable file.
**VWPProgramTemplateFunctionName*	The name of the entry point in the DLL that the Data Warehouse Center is to invoke if the value of **VWPProgramTemplateType* is ISV_PROGRAMTYPEDLL. This token is required if the value of **VWPProgramTemplateType* is ISV_PROGRAMTYPEDLL.	A text string, up to 80 bytes in length.
	Relationship parameters	

Table 48. VWPProgramTemplate.tag tokens (continued)

Token	Description	Allowed values
**VWPGroup*	The name of the group that is to contain the Data Warehouse Center program. This token is required.	A text string, up to 80 bytes in length.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.
**AgentSite*	The agent site to use for the source or target. This token is required.	A text string, up to 80 bytes in length. Specify `ISV_DEFAULTAGENTSITE` for the default agent site.

Examples of values

Table 49 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

Table 49. Example values for VWPProgramTemplate.tag tokens

Token	Example value
**VWPProgramTemplateName*	My ISV program
**VWPProgramTemplateDescription*	This program exports data from an ODBC database.
**VWPProgramTemplateNotes*	This program will export data from an ODBC database, process it, and place it into another database.
**VWPProgramTemplateExecutableName*	c:\ISV\BIN\MYPROG.EXE
**VWPProgramTemplateType*	ISV_PROGRAMTYPEEXECUTABLE
**VWPProgramTemplateFunctionName*	My_Prog_Func_Name
**VWPGroup*	Group of programs for partner tool
**CurrentCheckPointID++*	3

VWPProgramTemplateParameter.tag

Use this template to define a parameter that the Data Warehouse Center is to pass to a Data Warehouse Center program.

VWPProgramTemplateParameter.tag

This template is required if the Data Warehouse Center program requires that the Data Warehouse Center pass parameters to it. You can specify that multiple parameters are passed to the Data Warehouse Center program by including this template for each parameter.

Use this template with the VWPProgramTemplate.tag file ("VWPProgramTemplate.tag" on page 93). This template defines the relationship between the parameter and its Data Warehouse Center program definition (VWPProgramTemplate.tag).

Tokens

Table 50 provides information about each token in the template.

Table 50. VWProgramTemplateParameter.tag tokens

Token	Description	Allowed values
	Entity parameters	
**VWPProgramTemplateParameterName*	The name or description of a parameter that is to be passed to a Data Warehouse Center program. The name must be unique within the Data Warehouse Center program. This token is required.	A text string, up to 80 bytes in length.
**VWPProgramTemplateParameterOrder*	A number, starting with 0, that indicates the order of the parameter in the parameter list. This token is required.	A numeric value.
**VWPProgramTemplateParameterData*	The data that is passed to the Data Warehouse Center program as the value of the parameter. This token is required.	A text string or a numeric value up to 240 bytes in length.

Table 50. VWProgramTemplateParameter.tag tokens (continued)

Token	Description	Allowed values
**VWPProgramTemplateParameterKey*	A key that uniquely identifies this program parameter template. The key must be unique from all other keys in the interchange file. **Tip:** Finish processing the VWPProgramTemplateParameter.tag template before increasing the value of the key. This token is required.	A numeric value.
**VWPProgramInstanceParameterType*	The type of value that this parameter contains. For example, character, numeric, or password data.	One of the following values: **ISV_ParameterTypeNone** The parameter type is unknown or not applicable. **ISV_ParameterTypeCharacter** The parameter type is character. **ISV_ParameterTypeNumeric** The parameter type is numeric. **ISV_ParameterTypePassword** The parameter type is password.
	Relationship parameters	
**VWPProgramTemplateName*	The name of the Data Warehouse Center program that is to use this parameter. This token is required.	A text string, up to 80 bytes in length.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 51 on page 98 provides example values for each token to illustrate the kind of metadata that you might provide for each token.

VWPProgramTemplateParameter.tag

Table 51. Example values for VWPProgramTemplateParameter.tag tokens

Token	Example value
**VWPProgramTemplateParameterName*	DB2 UDB user ID
**VWPProgramTemplateParameterOrder*	1
**VWPProgramInstanceKey*	070000
**VWPProgramTemplateParameterData*	my_userid
**VWPProgramTemplateParameterKey*	012994
**VWPProgramInstanceParameterType*	ISV_ParameterTypePassword
**VWPProgramTemplateName*	My ISV program
**CurrentCheckPointID++*	4

WarehouseDataBase.tag

Use this template to define target warehouse databases to import into the Data Warehouse Center.

This template also defines the relationship between the following objects:

- The target warehouse database
- The agent site to use for the target warehouse database
- The security group in which to define the target warehouse database

Tokens

Table 52 provides information about each token in the template.

Table 52. WarehouseDataBase.tag tokens

Token	Description	Allowed values
	Entity parameters	
**DatabaseName*	The unique name of the database. The name must be unique within the warehouse control database. This token is required.	A text string, up to 80 bytes in length.
**DatabaseDescription*	The short description of the database. This token is optional.	A text string, up to 254 bytes in length.

Table 52. WarehouseDataBase.tag tokens (continued)

Token	Description	Allowed values
**DatabaseNotes*	The long description of the database. This token is optional.	A text string, up to 32700 bytes in length.
**DatabaseContact*	The person to contact for information about this database. This token is optional.	A text string, up to 64 bytes in length.
**DatabaseServerName*	The name of the server on which the database resides. This token is optional.	A text string, up to 64 bytes in length.
**DatabaseVersion*	The version of the database.	A text string.
**DatabasePhysicalName*	The physical database name of the database as defined to the database manager. This token is required.	A text string, up to 40 bytes in length.

Table 52. WarehouseDataBase.tag tokens (continued)

Token	Description	Allowed values
**DatabaseType*	The type of database family. This token is required.	One of the following values: **ISV_IR_DB2Family** DB2 Family **ISV_IR_Oracle** Oracle **ISV_IR_Sybase** Sybase **ISV_IR_MSSQLServer** Microsoft SQLServer **ISV_IR_Informix** Informix **ISV_IR_GenericODBC** Generic ODBC **ISV_IR_FFLan** Flat File LAN **ISV_IR_VSAM** VSAM **ISV_IR_IMS** IMS
**DatabaseTypeExtended*	The type of AS/400 system or file. This token is required.	One of the following values: **ISV_IR_DB2400CISC** DB2 UDB for AS/400 for CISC **ISV_IR_DB2400RISC** DB2 UDB for AS/400 for RISC **ISV_IR_FFLanLocalCmd** Local flat file **ISV_IR_FFLanFTPCopy** Local flat file sent using FTP from a remote system
**DatabaseUserid*	The user ID with which to access the database. This token is optional.	A text string, up to 36 bytes in length.
	Relationship parameters	

Table 52. WarehouseDataBase.tag tokens (continued)

Token	Description	Allowed values
**SecurityGroup*	The security group in which to create the source or target database. This token is required, but you can specify the default security group.	A text string, up to 80 bytes in length. Specify `ISV_DEFAULTSECURITYGROUP` for the default security group.
**AgentSite*	The agent site to use for the source or target. This token is required.	A text string, up to 80 bytes in length. Specify `ISV_DEFAULTAGENTSITE` for the default agent site.
**CurrentCheckPointID++*	An index, starting with 0, that increases each time that it is substituted in a token. This token is required.	A numeric value.

Examples of values

Table 53 provides examples of values for each token to illustrate the kind of metadata that you might provide for each token.

Table 53. example values for WarehouseDataBase.tag tokens

Token	Example value
**DatabaseName*	Finance Warehouse
**DatabaseDescription*	This database contains financial information.
**DatabaseNotes*	This is the warehouse where all geographies keep financial information.
**DatabaseContact*	Valerie Zieman
**DatabaseServerName*	CHI11W71
**DatabaseVersion*	V6.1.0
**DatabasePhysicalName*	FINANCE
**DatabaseType*	DB2 Family
**DatabaseTypeExtended*	ISV_DEFAULTVALUE
**DatabaseUserid*	DB2ADMIN
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**AgentSite*	My agent site

Table 53. example values for WarehouseDataBase.tag tokens (continued)

Token	Example value
**CurrentCheckPointID++*	6

Chapter 6. Data Warehouse Center metadata

This chapter describes the Data Warehouse Center metadata that describes source databases and target databases. Other applications can export the metadata to share information about the databases.

Table 54 describes the mapping between each object in the tag language file and the corresponding logical object in the Data Warehouse Center.

Table 54. Logical objects for source and target databases

Object in tag language file	Data Warehouse Center logical object	See:
DATABASE	A warehouse source or warehouse target	"DATABASE object"
TABLE	A table, file, or IMS segment	"TABLES object" on page 108
COLUMN	A column or field	"COLUMN object" on page 114

The Data Warehouse Center also defines relationships between the database, tables, and columns. The section for each object lists the relationships in which the object participates that are useful for partner applications.

DATABASE object

The DATABASE object contains metadata about a source database or target database, file system, or file.

Properties

Table 55 provides information about the properties of the DATABASE object.

Table 55. Properties of the DATABASE object

Tag language property name	Description	Allowed values
NAME	The business name of the source.	A text string, up to 80 bytes in length.

Table 55. Properties of the DATABASE object (continued)

Tag language property name	Description	Allowed values
DBNAME	The physical database name as defined to the database manager. This value is null for generic ODBC databases, Sybase databases, IMS databases, generic ODBC databases, and file systems.	A text string, up to 40 bytes in length.
SHRTDESC	The short description of the source.	A text string, up to 200 bytes in length.
LONGDESC	The long description of the source.	A text string, up to 32700 bytes in length.
DBTYPE	The database or file family.	One of the following values: **1** DB2 Family **20** Oracle **30** Sybase **40** Microsoft SQLServer **50** Informix **60** Generic ODBC **70** Flat File LAN **80** VSAM **90** IMS

Table 55. Properties of the DATABASE object (continued)

Tag language property name	Description	Allowed values	
DBETYPE	The type of database or file within a family.	One of the following values:	
		1	DB2/2
		3	DB2 MVS
		4	AS/400 CISC
		5	AS/400 RISC
		6	DB2/6000
		8	DB2 HP
		9	DB2 SUN
		11	DB2 NT
		12	DB2 VM
		13	DB2 SINIX
		14	DB2 SCO
		15	DB2 VSE
		16	DB2 EEE
		18	DB2 family
		19	DataJoiner
		20	Oracle
		30	Sybase
		40	Microsoft SQLServer
		50	Informix
		60	User-defined ODBC
DBETYPE (continued)	The type of database or file within a family.	One of the following values:	
		70	Flat File LAN Local Command
		71	Flat File LAN FTP Copy
		80	VSAM
		90	IMS

Table 55. Properties of the DATABASE object (continued)

Tag language property name	Description	Allowed values
ISWH	A flag that indicates whether this source is a warehouse target or warehouse source.	One of the following values: **Y** This source is a warehouse target. **N** This source is a warehouse source.
USERID	The user ID that the Data Warehouse Center uses to connect to the source.	A text string, up to 36 bytes in length.
CONTACT	The name of the person who is responsible for the source.	A text string, up to 64 bytes in length.
USEODBC	A flag that specifies whether to use the user-supplied connect string or to generate the string. Use N for files.	One of the following values: **Y** Use the user-defined connect string. **N** Generate the connect string.
ODBCSTR	The user-defined ODBC connect string to use if USEODBC is set to Y. Otherwise, this property is null.	A text string, up to 254 bytes in length.
PREACCMD	If the source is a local Flat File LAN source, a command to run to access the remote file.	A text string, up to 64 bytes in length.
POSTACMD	If the source is a local Flat File LAN source, a command to run after accessing the remote file.	A text string, up to 64 bytes in length.
RETRYCNT	The number of times to try to extract data from this source in case of an error.	A numeric value.
RETRYINT	The time that is to elapse between attempts to extract data.	A numeric value.
VERSION	The version of DB2 in use.	A text string, up to 128 bytes in length.
DBMSSERV	The database instance/subsystem/server name for ODBC connect.	A text string, up to 128 bytes in length.
DFLTDEL	The System 390 database default character string delimiter.	A text string, up to 1 byte in length.

Figure 13 on page 107 shows an example of a DATABASE object instance that defines a target warehouse database.

```
:COMMENT.  Begin DATABASE Instance
:COMMENT.
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(DATABASE)
:INSTANCE.
    NAME(iwhtar)
    DBNAME(IWHTAR)
    DBTYPE(1)
    DBETYPE(11)
    ISWH(Y)
    USERID(marlow)
    USEODBC(N)
    CODEPAGE(437)
    RETRYCNT(3)
    RETRYINT(30)
```

Figure 13. Target DATABASE object instance

Figure 14 shows an example of a DATABASE object instance that defines a source file.

```
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(DATABASE)
:INSTANCE.
    NAME(TBC Operations)
    SHRTDESC(The Beverage Company operational data sources)
    DBTYPE(70)
    DBETYPE(70)
    ISWH(N)
    LOCATION(Thirsty City)
    USERID(XXXXXXXX)
    USEODBC(N)
    CODEPAGE(437)
    RETRYCNT(0)
    RETRYINT(0)
```

Figure 14. Source file DATABASE object instance

Relationships

Table 56 on page 108 shows the relationship in which the DATABASE object participates and that is useful for partner applications. The Source column and the Target column indicate how many times the source object or the target object of the relationship can participate in the relationship. For example, in Table 56 on page 108, the values 1 and M indicate that one database can relate to many tables, but a table can relate only to one database.

DATABASE object

Table 56. Relationships in which the DATABASE object participates

Source	Source tag language object type	Relation type	Target	Target tag language object type	Description
1	DATABASE	CONTAIN	M	TABLES	Tables or files that are contained in the database or file system.

Figure 15 shows an example of a relationship between a DATABASE object instance and a TABLES object instance.

```
:COMMENT.  Relation: DATABASE to TABLES
:COMMENT.
:ACTION.RELATION(ADD)
:RELTYPE.TYPE(CONTAIN) SOURCETYPE(DATABASE) TARGETYPE(TABLES)
:INSTANCE.
    SOURCEKEY(NAME(TBC Operations) DBNAME() )
    TARGETKEY(DBNAME(TBC Operations) OWNER() TABLES(d:\iwhdemo\outcusti.txt) )
```

Figure 15. Linking DATABASE object instance to TABLES object instance

TABLES object

This object contains metadata about a warehouse source table, segment, or file, or a target table. It is associated with a DATABASE object (see “DATABASE object” on page 103).

Properties

Table 57 provides information about the properties of the TABLES object.

Table 57. Properties of the TABLES object

Tag language property name	Description	Allowed values
NAME	The name of the table, file, or IMS segment. The table name includes the high-level qualifier, schema or collection, such as IWH.TABLE1. The combination of the database name and the table name is unique. This property is the fully qualified path and file name for a file.	A text string, up to 80 bytes in length.

Table 57. Properties of the TABLES object (continued)

Tag language property name	Description	Allowed values
SHRTDESC	The short description of the file or segment.	A text string, up to 200 bytes in length.
LONGDESC	The long description of the table.	A text string, up to 32700 bytes in length.
DBNAME	The business name of the source that contains this table or file.	A text string, up to 80 bytes in length.
OWNER	The owner, high-level qualifier, or collection of the table. This property is null for files and IMS segments.	A text string, up to 15 bytes in length.
TABLES	The physical table, file, or segment name as defined to the database manager or file system. For files and IMS segments, this value is the same as the value of NAME.	A text string, up to 80 bytes in length.
TBLISBIN	A flag that specifies the file transfer mode for Flat File LAN files.	One of the following values: **Y** The file transfer mode is binary. **N** The file transfer mode is ASCII.
TBLNAMESP	The name of the DB2 table space.	A text string, up to 90 bytes in length.
TBLFTYPE	For files, the type of the file.	One of the following values: **1** Fixed **2** Comma **3** Tab **4** Character
TBLL1NAM	A flag that specifies whether the first row of the file contains column names.	One of the following values: **Y** The first row of the file contains column names. **N** The first row of the file contains data.

Table 57. Properties of the TABLES object (continued)

Tag language property name	Description	Allowed values
CHARDELM	For files, the character separator if the file type is character.	A text string that is 1 byte in length.
CREATYPE	The method used to define the table in the Data Warehouse Center.	One of the following values: **1** The table was defined manually. **2** The table definition was imported from the database manager. **3** The table definition was imported from the Information Catalog Manager. **4** The table was created by the Data Warehouse Center for a step when the step was promoted to test mode.
TABALIAS	A flag that specifies whether the table has an alias.	One of the following values: **Y** The table has an alias. **N** The table does not have an alias.
IWHCRTAR	A flag that specifies whether the target table is created by the Data Warehouse Center.	One of the following values: **Y** The target table is created by the Data Warehouse Center. **N** The target table is not created by the Data Warehouse Center.
IWHGRANT	A flag that specifies whether GRANT TO PUBLIC is enabled for the table.	One of the following values: **Y** GRANT TO PUBLIC is enabled for the table. **N** GRANT TO PUBLIC is been enabled for the table.

Table 57. Properties of the TABLES object (continued)

Tag language property name	Description	Allowed values
IWHDRATN	The warehouse target duration, either transient or persistent.	One of the following values: **Y** The table is persistent. **N** The table is transient.
IWHMAXED	The maximum number of editions of the table.	A numeric value.
IWHCREGN	A flag that specifies whether the create statement is automatically generated.	One of the following values: **Y** The Create statement is automatically generated. **N** The Create statement is not automatically generated.
IWHCRERU	The create statement for the table.	A text string, up to 32,700 bytes in length.
IDSFACT	A flag that specifies whether the table is used as a fact table.	One of the following values: **Y** The table is used as a fact table. **N** The table is not used as a fact table.
CDSSCHEMA	The table schema for replication.	A text string, up to 128 bytes in length.
CDTABNAM	The table name for replication.	A text string, up to 128 bytes in length.
BEFORIMG	The replication before-image prefix.	A text string, up to 4 bytes in length.
IDSREPL	A flag that specifies whether the table is used for replication.	One of the following values: **Y** The table is used for replication. **N** The table is not used for replication.
NAMINDEX	The DB2 table name index.	A text string, up to 90 bytes in length.

Table 57. Properties of the TABLES object (continued)

Tag language property name	Description	Allowed values
PARTTBSP	A flag that specifies whether the table is in a partitioned table space.	One of the following values: **Y** The table is in a partitioned table space. **N** The table is not in a partitioned table space.
DBNAM390	The System 390 database name.	A text string, up to 8 bytes in length.

Figure 16 shows an example of a TABLES object instance for a relational table.

```
:COMMENT.  Begin TABLES Instance
:COMMENT.
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(TABLES)
:INSTANCE.
    NAME(IWH.ATOMICED)
    DBNAME(iwhtar)
    OWNER(IWH)
    TABLES(ATOMICED)
    TBLISBIN(N)
    TBLFTYPE(0)
    TBLL1NAM(N)
    CREATYPE(4)
:COMMENT.
:COMMENT.  End TABLES Instance
```

Figure 16. TABLES object instance for a relational table

Figure 17 on page 113 shows an example of a TABLES object instance for a file.

```
:COMMENT.  Begin TABLES Instance
:COMMENT.
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(TABLES)
:INSTANCE.
    NAME(d:\iwhdemo\outcusti.txt)
    SHRTDESC(File containing operational data for Institutions Customers)
    DBNAME(TBC Operations)
    OWNER()
    TABLES(d:\iwhdemo\outcusti.txt)
    TBLISBIN(Y)
    TBLFTYPE(3)
    TBLL1NAM(N)
    CREATYPE(1)
:COMMENT.
:COMMENT.  End TABLES Instance
```

Figure 17. TABLES object instance for a file

Relationships

Table 58 lists the relationships in which the TABLES object participates and that are useful for partner applications. The Source column and the Target column indicate how many times the source object or target object of the relationship can participate in the relationship.

Table 58. Relationships in which the TABLES object participates

Source	Source tag language object type	Relation type	Target	Target tag language object type	Description
1	DATABASE	CONTAIN	M	TABLES	Database or file system with which this table or file is associated.
1	TABLE	CONTAIN	M	COLUMN	Columns associated with this table.

Figure 18 on page 114 shows an example of a relationship between a TABLES object instance and a DATABASE object instance.

```
:COMMENT.  Relation: DATABASE to TABLES
:COMMENT.
:ACTION.RELATION(ADD)
:RELTYPE.TYPE(CONTAIN) SOURCETYPE(DATABASE) TARGETYPE(TABLES)
:INSTANCE.
    SOURCEKEY(NAME(TBC Operations) DBNAME() )
    TARGETKEY(DBNAME(TBC Operations) OWNER() TABLES(d:\iwhdemo\outcusti.txt) )
```

Figure 18. Linking TABLES object instance to DATABASE object instance

Figure 19 shows an example of a relationship between a TABLES object instance and a COLUMN object instance.

```
:COMMENT.  Relation: TABLES to COLUMN
:COMMENT.
:ACTION.RELATION(ADD)
:RELTYPE.TYPE(CONTAIN) SOURCETYPE(TABLES) TARGETYPE(COLUMN)
:INSTANCE.
    SOURCEKEY(DBNAME(TBC Operations) OWNER() TABLES(d:\iwhdemo\outcusti.txt) )
    TARGETKEY(DBNAME(TBC Operations) OWNER() TABLES(d:\iwhdemo\outcusti.txt)
  COLUMNS(Zipcode) )
```

Figure 19. Linking TABLES object instance to COLUMN object instance

COLUMN object

The COLUMN object contains metadata about a column or field in a source table, target table, or file. It is associated with a TABLES object (see "TABLES object" on page 108).

Properties

Table 59 provides information about the properties of the COLUMN object.

Table 59. Properties of the COLUMN object

Tag language property name	Description	Allowed values
NAME	The name of the column or field. The combination of the database name, table name, and column name is unique.	A text string, up to 80 bytes in length.
SHRTDESC	The short description of the column or field.	A text string, up to 200 bytes in length.
LONGDESC	The long description of the column or field.	A text string, up to 32700 bytes in length.

Table 59. Properties of the COLUMN object (continued)

Tag language property name	Description	Allowed values
DATATYPE	The ODBC data type to which the database manager data type maps. The Data Warehouse Center derives the data type from the native data type. You cannot add a GRAPHIC data type column to a table in a VSAM database.	One of the following values: CHAR NUMERIC DECIMAL INTEGER SMALLINT FLOAT DOUBLE DATE TIME TIMESTAMP VARCHAR LONG_VARCHAR GRAPHIC VARGRAPHIC LONG_VARGRAPHIC BLOB CLOB DBCLOB TINYINT BIT REAL BIGINT
LENGTH	The length of the column or field.	A numeric value.
SCALE	The precision of the column or field for columns or fields with a decimal data type.	A numeric value.
POSNO	An index, starting with 0, of the column or field in the row of the table or file.	A numeric value.
NULLS	A flag that specifies whether the column or field allows null data.	One of the following values: **Y** The column allows null data. **N** The column does not allow null data.
ISTEXT	A flag that specifies whether the column or field data is binary or text data.	One of the following values: **Y** The column data is binary data. **N** The column data is text data.
DBNAME	The business name of the source or target that contains this table or file.	A text string, up to 80 bytes in length.

Table 59. Properties of the COLUMN object (continued)

Tag language property name	Description	Allowed values
OWNER	The owner, high-level qualifier, or collection of the table. This property is null for files and IMS segments.	A text string, up to 15 bytes in length.
TABLES	The physical table, file, or segment name as defined to the database manager or file system. For files and IMS segments, this value is the same as the value of NAME.	A text string, up to 80 bytes in length.
NATIVEDT	Native data type of the column or field.	The data type for the column as defined to the database manager. The data type is a text string, up to 40 bytes in length. In most cases, the value of this property will match the value of DATATYPE. For the mapping of the database manager data types to ODBC data types, see the Data Warehouse Center online help.
ORDINAL	Column or field ordinality.	A numeric value.
OFFSET	The offset of the field in a fixed-length file.	A numeric value.
COLTYPE	The column type for DPropR.	One of the following values: **A** After image column **B** Before image column

Figure 20 on page 117 shows an example of a COLUMN object instance.

```
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(COLUMN)
:INSTANCE.
    NAME(CORR_COEF)
    SHRTDESC(Correlation Coefficient)
    DATATYPE(DOUBLE)
    LENGTH(0)
    SCALE(0)
    POSNO(4)
    NULLS(Y)
    ISTEXT(N)
    DBNAME(TRANSFORMER_TARGET)
    OWNER(IWH)
    TABLES(TR_CORRELATION_06)
    COLUMNS(CORR_COEFF)
    NATIVEDT(DOUBLE)
    TRANSNAM(Correlation Coefficient(r))
```

Figure 20. COLUMN object instance

Relationships

Table 60 shows the relationship in which the COLUMN object participates. This relationship is useful for partner applications. The Source column and the Target column indicate how many times the source object or the target object of the relationship can participate in the relationship.

Table 60. Relationship in which the COLUMN object participates

Source	Source tag language object type	Relation type	Target	Target tag language object type	Description
1	TABLES	CONTAIN	M	COLUMN	The table with which this column is associated.

Figure 21 on page 118 shows an example of a relationship between a COLUMN object instance and a TABLES object instance.

```
:COMMENT.  Relation: TABLES to COLUMN
:COMMENT.
:ACTION.RELATION(ADD)
:RELTYPE.TYPE(CONTAIN) SOURCETYPE(TABLES) TARGETYPE(COLUMN)
:INSTANCE.
    SOURCEKEY(DBNAME(TBC Operations) OWNER() TABLES(d:\iwhdemo\outcusti.txt) )
    TARGETKEY(DBNAME(TBC Operations) OWNER() TABLES(d:\iwhdemo\outcusti.txt)
     COLUMNS(Zipcode) )
```

Figure 21. Linking COLUMN object instance to TABLES object instance

Chapter 7. Information Catalog Manager system tables and metadata models

The following tables are defined for Information Catalog Manager system usage:

- Attachment Relation table: FLG.ATCHREL
- Check Point Working table: FLG.CHECKPT
- Comments table: FLG.COMMENTS
- Exchange table: FLG.EXCHANGE
- History table: FLG.HISTORY
- Object Name Instance table: FLG.NAMEINST
- Object Type Register table: FLG.OBJTYREG
- Long Description Overflow table: FLG.OVERDESC
- System Parameter table: FLG.PARMS
- Programs table: FLG.PROGRAMS
- Object Type Property table: FLG.PROPERTY
- Relation Instance table: FLG.RELINST
- Users table: FLG.USERS
- Windows Icons table: FLG.WINICON

FLG.ATCHREL table

The FLG.ATCHREL table is used to define a relationship between an object instance and a comment.

The RELTYPE, SOURCE, and TARGET columns form the primary key of table.

The RELTYPE column is an index of the table.

Table 61 on page 120 provides information about each column found in the FLG.ATCHREL table.

Table 61. FLG.ATCHREL table column properties

Column name	Data type	Description	Nullable	NLS
RELTYPE	CHAR(1)	Relation type: **A** Attachment relation; **L** Link relation; **M** Comments relation	No	SBCS
SOURCE	CHAR(16)	The FLGID that represents the source object instance.	No	SBCS
TARGET	CHAR(16)	The FLGID that represents the target object instance	No	SBCS
Note: NLS: National Language Support; SBCS: Single Byte Character Set; DBCS: Double Byte Character Set				

FLG.CHECKPT table

The FLG.CHECKPT table is used by the Import API to restart the import process at a checkpoint.

The table is populated by the Import API. At any time, this table can contain zero to many rows.

The TAGFNAME column is the primary key of table.

The COMMITID, LASTUPDT, and USERID columns are all indexes of the table.

Table 62 provides information about each column found in the FLG.CHECKPT table.

Table 62. FLG.CHECKPT table column properties

Column name	Data type	Description	Nullable	NLS
TAGFNAME	VARCHAR(240)	The name of the tag language file (without the path information).	No	Both SBCS and DBCS

Table 62. FLG.CHECKPT table column properties (continued)

Column name	Data type	Description	Nullable	NLS
COMMITID	CHAR(26)	The identifier of the last COMMIT checkpoint. This identifier is supplied by the user in a COMMIT tag placed at appropriate locations in the tag language file. It can be a system timestamp or any series of characters.	No	Both SBCS and DBCS
LASTUPDT	TIMESTAMP	The system timestamp when this entry was either created or updated. The Last Update field will not need padding, because it will always occupy the full 26 bytes.	No	None
USERID	CHAR(8)	The user ID of the information catalog administrator.	No	Both SBCS and DBCS
ENTSAVED	INTEGER	The total number of entries that have been saved in the save area.	No	None
SAVEAREA	LONG VARCHAR	Storage area for a list of object type names. Each object type name is 8 bytes.	No	SBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.COMMENTS table

The FLG.COMMENTS table contains all the comments on objects in the Information Catalog Manager information catalog.

At any time, this table may contain zero to many rows.

The INSTIDNT column is the primary key of the table.

The NAME, CREATOR, and CREATSTP columns form the unique index of the table.

The NAME, CREATOR, CREATSTP, and UPDATIME columns are indexes of the table.

Table 63 on page 122 provides information about each column found in the FLG.COMMENTS table.

Table 63. FLG.COMMENTS table column properties

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	This six-digit object type ID, generated by the Information Catalog Manager, represents a specific object type in the information catalog.	No	SBCS
INSTIDNT	CHAR(10)	The unique instance ID generated by the Information Catalog Manager. It is the second part of the FLGID, the 10-digits serial number that will uniquely identify this instance within its own object type.	No	SBCS
NAME	VARCHAR(80)	The name entered by the information catalog user to identify each user-defined object instance.	No	Both SBCS and DBCS
UPDATIME	CHAR(26)	The date and time of the metadata creation or last update. This date is generated by the Information Catalog Manager.	Yes	None
UPDATEBY	CHAR(8)	The user ID of the information catalog administrator who last updated the instance.	Yes	Both SBCS and DBCS
CREATOR	CHAR(8)	The creator of the Comments object. The system will set the creator value to the current user ID.	No	Both SBCS and DBCS
CREATSTP	CHAR(26)	A timestamp indicating the date and time the Comments object instance was created. This timestamp is supplied by the system when the instance is created.	No	None
STATUS	CHAR(80)	The status of the comment. Users can design their own conventions for this value.	Yes	Both SBCS and DBCS
ACTIONS	VARCHAR(250)	Specifies what action the user should take.	Yes	Both SBCS and DBCS
EXTRA	VARCHAR(80)	Used for extra information.	Yes	Both SBCS and DBCS
Note: NLS: National Language Support; SBCS: Single Byte Character Set; DBCS: Double Byte Character Set				

FLG.EXCHANGE table

The FLG.EXCHANGE table is used to keep track of the object sychronized between the Information Catalog Manager, the Data Warehouse Center, and DB2 OLAP Server™.

This table is populated by the metadata interchange at installation time.

The OBJNAME and OBJTYPE columns form the primary key of the table.

Table 64 provides information about each column found in the FLG.EXCHANGE table.

Table 64. FLG.EXCHANGE table column properties

Column name	Data type	Description	Nullable	NLS
PRODUCT	VARCHAR(40)	The combination of product, version, and release numbers.	No	SBCS
OBJNAME	VARCHAR(200)	The object name, for example, step.	No	Both SBCS and DBCS
IMPDATE	TIMESTAMP	The import timestamp.	No	None
OBJTYPE	CHAR(5)	OBJTYPE can be one of the following values: • IR represents source metadata exchanged • DR represents target metadata • BV represents step metadata • OLAP represents OLAP metadata	No	SBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.HISTORY table

The FLG.HISTORY table is used to keep track of object instances that have been deleted from the Information Catalog Manager and the Data Warehouse Center.

The table is populated when the user deletes an object instance and the recording delete history flag is ON. At any time, this table can contain zero to many rows.

The HISSEQ column is the primary key of the table.

Table 65 provides information about each column found in the FLG.HISTORY table.

Table 65. FLG.HISTORY table column properties

Column name	**Data type**	**Description**	**Nullable**	**NLS**
HISSEQ	TIMESTAMP	The sequence number of the delete history.	No	None
HISTYPE	INTEGER	The type of the delete history. • A value of 1 in this column indicates a deletion from the information catalog. • A value of 2 in this column indicates a deletion from the Data Warehouse Center.	No	None
HISTAG	LONG VARCHAR	This column will store the identifier of the object to be deleted.	Yes	Both SBCS and DBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.NAMEINST table

The FLG.NAMEINST table contains the name of every object in the information catalog.

The FLGID column is the primary key of the table.

The INSTNAME and TYPENAME columns are indexes of the table.

Table 66 provides information about each column found in the FLG.NAMEINST table.

Table 66. FLG.NAMEINST table column properties

Column name	**Data type**	**Description**	**Nullable**	**NLS**
FLGID	CHAR(16)	The 16-character object instance ID.	No	SBCS
TYPENAME	VARCHAR(80)	The external name of the object type.	No	Both SBCS and DBCS

Table 66. FLG.NAMEINST table column properties (continued)

Column name	Data type	Description	Nullable	NLS
INSTNAME	VARCHAR(80)	The external name of an object instance.	No	Both SBCS and DBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.OBJTYREG table

The FLG.OBJTYREG table is used to keep track of all objects and their object types, as well as tables created by the Information Catalog Manager.

The OBJTYPID column is the primary key of FLG.OBJTYREG that uniquely identifies an object type in the information catalog and is used as the prefix for all instance IDs.

The columns PTNAME, NAME, and DPNAME are unique index keys of the FLG.OBJTYREG table.

The columns CATEGORY, CREATOR, and UPDATEBY are index keys of the table.

Table 67 provides information about each column found in the FLG.OBJTYREG table.

Table 67. FLG.OBJTYREG table column properties

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	The six-digit object type ID generated by the Information Catalog Manager. The ID represents a specific object type in the information catalog.	No	SBCS
PTNAME	CHAR(30)	The name of the object type. The name is also used as the name of the user's table. The actual size of PTNAME is determined by the value of ENVSIZE on the FLG.PARMS table, which is defined during installation.	No	SBCS
DPNAME	CHAR(8)	The unique object type name within an information catalog.	No	SBCS

Table 67. FLG.OBJTYREG table column properties (continued)

Column name	Data type	Description	Nullable	NLS
NAME	VARCHAR(80)	The external name of this object type.	No	Both SBCS and DBCS
CATEGORY	CHAR(1)	The Information Catalog Manager categories: Elemental E, Grouping G, Program P, Contact C, Dictionary D, Support S, and Attachment A.	No	SBCS
CREATOR	CHAR(8)	The user ID of the information catalog administrator who created the object type. It will be blank when the object type is registered. It will also contain a blank after the object type is deleted but before the registration is removed.	Yes	Both SBCS and DBCS
UPDATIME	CHAR(26)	The date and time of the object type that was created or that had its properties extended.	Yes	SBCS
UPDATEBY	CHAR(8)	The user ID of the information catalog administrator who last extended the object type (appended properties).	Yes	Both SBCS and DBCS
LASTINID	INTEGER	The last system-generated instance ID for this object type. This is an internal property, and it will not be visible to the information catalog user. It is accessed and updated by the Create Instance IPI only.	No	None
OBJICON	LONG VARCHAR FOR BIT DATA	The icon bitmap corresponding to the object type.	No	None
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.OVERDESC table

The FLG.OVERDESC table contains all long description properties. Each long description is divided into 3-KB chunks.

The OBJTYPID, INSTIDNT, PHYPRPNM, and SEQNO columns form the primary key of table FLG.OVERDESC.

Table 68 provides information about each column found in the FLG.OVERDESC table.

Table 68. FLG.OVERDESC table column properties

Column name	**Data type**	**Description**	**Nullable**	**NLS**
OBJTYPID	CHAR(6)	The six-digit, object type ID generated by the Information Catalog Manager, represents a specific object type in the information catalog.	No	SBCS
INSTIDNT	CHAR(10)	The unique instance ID generated by the Information Catalog Manager. The ID is the second part of the FLGID, the 10-digit portion of the serial number that uniquely identifies this instance within its own object type.	No	SBCS
PHYPRPNM	CHAR(8)	The original property or column name defined by the user.	No	SBCS
SEQNO	SMALLINT	A sequence number to keep track of how many rows reflect the same incoming source.	No	None
ODESC	VARCHAR(3000)	This entry keeps the segments of a long description, which can be up to 32700 bytes, in a smaller and more manageable buffer.	No	Both SBCS and DBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.PARMS table

The FLG.PARMS table does not contain metadata. It contains internal, global parameters for the Information Catalog Manager. The table is a global storage area for persistent Information Catalog Manager parameters such as version, logon message, and code page.

FLG.PARMS stores system parameters. The values in this table are set when you use the Information Catalog Manager Create Catalog Utility (see *Information Catalog Manager Administration Guide*). You can also use the Information Catalog Manager APIs (see the *Information Catalog Manager Programming Guide and Reference*) to change the values.

Table 69 provides information about each column found in the FLG.PARMS table.

Table 69. FLG.PARMS table column properties

Column name	Data type	Description	Nullable	NLS
VERSION	CHAR(20)	The version of the information catalog, for example, V1R0M0 or V1R1M0; which is populated at the installation or migration time.	Yes	SBCS
LOGONMSG	VARCHAR(254)	Information Catalog Manager logon message, for example, "Welcome to the Information Catalog Manager!"	Yes	Both SBCS and DBCS
CODEPAGE	CHAR(4)	Code page number of the information catalog.	Yes	SBCS
LANGUAGE	CHAR(4)	Language code, for example, ENU (US English). It is loaded from a string file.	Yes	SBCS
DTOKEN	CHAR(1)	The default token of the Information Catalog Manager environment used to represent an unspecified data field. This not-applicable symbol is used by the import and export functions. This value is set during installation.	Yes	SBCS
ENVSIZE	SMALL INTEGER	Database server environment size. This value is set during installation, and is used to specify the proper name length for Information Catalog Manager tables, columns, and indexes. This value can be 10 for DB2 UDB for AS/400, 18 for most other IBM relational databases, and up to a maximum of 30 bytes for non-IBM databases.	Yes	None
LASTYPID	INTEGER	The last system-generated ID for an object type. The ID is accessed and updated by the Create Registration IPI only.	Yes	None
LISTMAX	INTEGER	The maximum number of retrievable objects from a listing or search result.	Yes	None
ISTGROUP	CHAR(8)	The index storage group name for the DB2 for OS/390® database.	Yes	SBCS
TSTGROUP	CHAR(8)	The table storage group name for the DB2 for OS/390 database.	Yes	SBCS
MDBNAME	CHAR(8)	The DB2 for OS/390 database name.	Yes	SBCS

Table 69. FLG.PARMS table column properties (continued)

Column name	Data type	Description	Nullable	NLS
TBSPAC32	CHAR(8)	The 32 KB table space name for the DB2 for OS/390 database.	Yes	SBCS
TBSPAC04	CHAR(8)	The 4 KB table space name for DB2 for OS/390 database.	Yes	SBCS
PARMFLAG	INTEGER	A flag indicator. **FLG_PARMS_RECORD_DELETE_HISTORY** Records the delete history. **FLG_PARMS_MVS_FOLD_UP** Saves the object values in uppercase in the DB2 for OS/390 information catalog. You can search these values in uppercase or lowercase in the Information Catalog Manager.	Yes	None
CMTSTAT	VARCHAR(800)	This column stores a list of comments status. Each status is 80 bytes.	Yes	Both SBCS and DBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.PROGRAMS table

The FLG.PROGRAMS table is used to keep track of all program objects in the information catalog.

The INSTIDNT column is the primary key of the table FLG.PROGRAMS.

The UUICLASS, UUIQUAL1, UUIQUAL2, UUIQUAL3, and UUIDENT columns form the unique index of table FLG.PROGRAMS.

The NAME, UPDATEBY, UPDATIME, UUICLASS, UUIQUAL1, UUIQUAL2, UUIQUAL3, UUIDENT, and HANDLES columns are indexes of the table.

Table 70 on page 130 provides information about each column found in the FLG.PROGRAMS table.

Table 70. FLG.PROGRAMS table column properties

Column name	Data type	Description	Origin	NLS
OBJTYPID	CHAR(6)	The six-digit object type ID, generated by the Information Catalog Manager, represents a specific object type.	No	SBCS
INSTIDNT	CHAR(10)	The unique instance ID generated by the Information Catalog Manager. It is the second part of the FLGID, the 10-digit serial number that uniquely identifies this instance within its own object type.	No	SBCS
NAME	VARCHAR(80)	This name is entered by the information catalog user to identify each user-defined object instance.	No	Both SBCS and DBCS
UPDATIME	CHAR(26)	The date and time of metadata creation or last update. This is generated by the Information Catalog Manager.	Yes	SBCS
UPDATEBY	CHAR(8)	The user ID of the information catalog administrator who last updated the instance.	Yes	Both SBCS and DBCS
UUICLASS	CHAR(25)	The part1 name of the universal unique identifier (UUI).	No	Both SBCS and DBCS
UUIQUAL1	VARCHAR(48)	The part2 name of the (UUI).	No	Both SBCS and DBCS
UUIQUAL2	VARCHAR(48)	The part3 name of the (UUI).	No	Both SBCS and DBCS
UUIQUAL3	VARCHAR(48)	The part4 name of the (UUI).	No	Both SBCS and DBCS
UUIDENT	VARCHAR(70)	The part5 name of the (UUI).	No	Both SBCS and DBCS
HANDLES	CHAR(8)	The object type that this program handles.	Yes	SBCS
STARTCMD	VARCHAR(250)	The program name to be invoked. The program can have an extension of .exe, .cmd, .com, or .bat.	No	Both SBCS and DBCS
PARMLIST	VARCHAR(1800)	If a parameter list is required to handle object instances, the value of the parameter is specified by the HANDLES property.	Yes	Both SBCS and DBCS

Table 70. FLG.PROGRAMS table column properties (continued)

Column name	Data type	Description	Origin	NLS
SHRTDESC	VARCHAR(250)	The short description of the program.	Yes	Both SBCS and DBCS
Note: NLS: National Language Support; SBCS: Single Byte Character Set; DBCS: Double Byte Character Set				

FLG.PROPERTY table

The FLG.PROPERTY table is used to define a property for an object type. There is one row for each property of each object type defined in this table. For a description of Information Catalog Manager object types and object type properties, see "Chapter 8. Information Catalog Manager object types" on page 147.

The OBJTYPID column is the index of the table.

Table 71 provides information about each column found in the FLG.PROPERTY table.

Table 71. FLG.PROPERTY table column properties

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	System-generated ID that is a unique 6 digits for each object type.	No	SBCS
PHYPRPNM	CHAR(8)	The physical name of the property in the object type. This name will be used to generate the column name in the user's object table.	No	SBCS
PROPNAME	VARCHAR(80)	The external name of this object type property.	No	Both SBCS and DBCS
DATATYPE	CHAR(30)	Property data type, CHAR, VARCHAR, LONG VARCHAR and TIMESTAMP.	No	SBCS
LENGTH	INTEGER	Property length.	No	None

Table 71. FLG.PROPERTY table column properties (continued)

Column name	Data type	Description	Nullable	NLS
OPTIONS	CHAR(1)	A value flag used to indicate if this field allows null values. **R** Value required (not nullable) **O** Optional value (nullable) **S** System generated value	No	SBCS
UUISEQNO	CHAR(1)	The UUI sequence number of the property in the object type.	Yes	SBCS
PROPSEQ	INTEGER	The sequence number of the property	No	None
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.RELINST table

The FLG.RELINST table defines relationships between two objects. The table contains one row for each source-to-target object instance relationship.

The RELTYPE, SOURCE, and TARGET columns form the primary key of the table.

The RELTYPE, SRCCAT, SOURCE, SRCTNAME, SRCINAME, TRGCAT, TARGET, TRGTNAME, and TRGINAME columns are indexes of the table.

Table 72 provides information about each column found in the FLG.RELINST table.

Table 72. FLG.RELINST table column properties

Column name	Data type	Description	Nullable	NLS
RELTYPE	CHAR(1)	Relation type: **C** Contains **T** Contact	No	SBCS
SRCCAT	CHAR(1)	Category of the source object.	No	SBCS

Table 72. FLG.RELINST table column properties (continued)

Column name	Data type	Description	Nullable	NLS
SOURCE	CHAR(16)	The FLGID that represents the source object instance.	No	SBCS
SRCTNAME	VARCHAR(80)	The external name of the source object type.	No	Both SBCS and DBCS
SRCINAME	VARCHAR(80)	The external name of the source object instance.	No	Both SBCS and DBCS
TRGCAT	CHAR(1)	The category of the target object.	No	SBCS
TARGET	CHAR(16)	The FLGID that represents the target object instance.	No	SBCS
TRGTNAME	VARCHAR(80)	The external name of the target object type.	No	Both SBCS and DBCS
TRGINAME	VARCHAR(80)	The external name of the target object instance.	No	Both SBCS and DBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.USERS table

The FLG.USERS table contains a list of all the information catalog administrators and users with special administrative privileges. Unlike most of the other Information Catalog Manager store tables, the FLG.USERS table does not contain metadata. It contains definitions of different types of information catalog users and their status.

The USERTYPE and DGUSER columns form the primary key of the table.

The DGUSER column is an index of the table.

Table 73 on page 134 provides information about each column found in the FLG.USERS table.

Table 73. FLG.USERS table column properties

Column name	Data type	Description	Nullable	NLS
DGUSER	CHAR(8)	The user ID of the information catalog administrator. The ID is entered at installation.	No	Both SBCS and DBCS
USERTYPE	CHAR(1)	Type of DGUSER. The type can be an information catalog administrator, a user with special update privileges, or a user. This value is set during installation.	No	SBCS
ACTIVEKA	CHAR(1)	A flag to indicate the information catalog administrator who is currently logged on to the Information Catalog Manager. Only one information catalog administrator can be logged on at a time.	Yes	SBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

FLG.WINICON table

The FLG.WINICON table contains the associated Windows icon for each object type.

The OBJTYPID column is the primary key of the table.

Table 74 provides information about each column found in the FLG.WINICON table.

Table 74. FLG.WINICON table column properties

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	The six-character object type ID.	No	SBCS
OBJICON	LONG VARCHAR FOR BIT DATA (30000)	The bitmap for the Windows icon.	Yes	None

Table 74. FLG.WINICON table column properties (continued)

Column name	Data type	Description	Nullable	NLS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

Information Catalog Manager metadata models

The following sections describe the Information Catalog Manager metadata models. "Model for Information Catalog Manager system tables" describes the relationships between Information Catalog Manager system tables. "Logical metadata model" on page 139 describes the relationships between objects in the Information Catalog Manager object type categories.

Model for Information Catalog Manager system tables

The following illustrations show the relationships between the different Information Catalog Manager system tables as well as the object-type tables. For example, a relationship can be a join between two columns. The following Information Catalog Manager system tables are not related to the other system tables:

- FLG.PARMS
- FLG.HISTORY
- FLG.USERS
- FLG.EXCHANGE
- FLG.CHECKPT

See the notes following this figure for each numbered relationship.

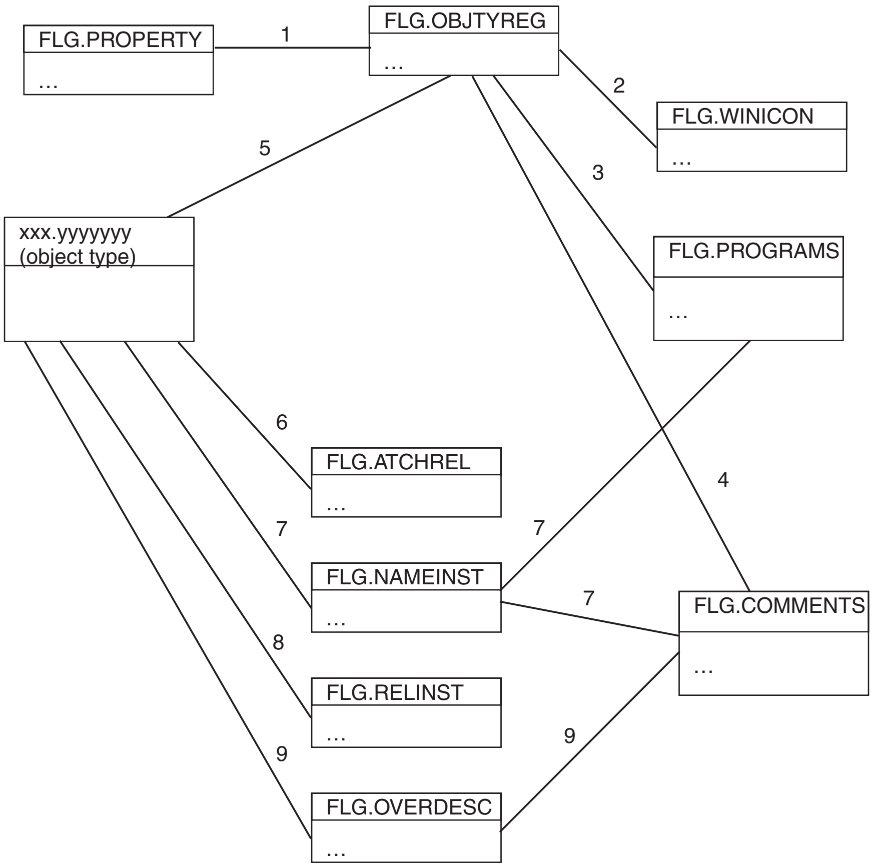
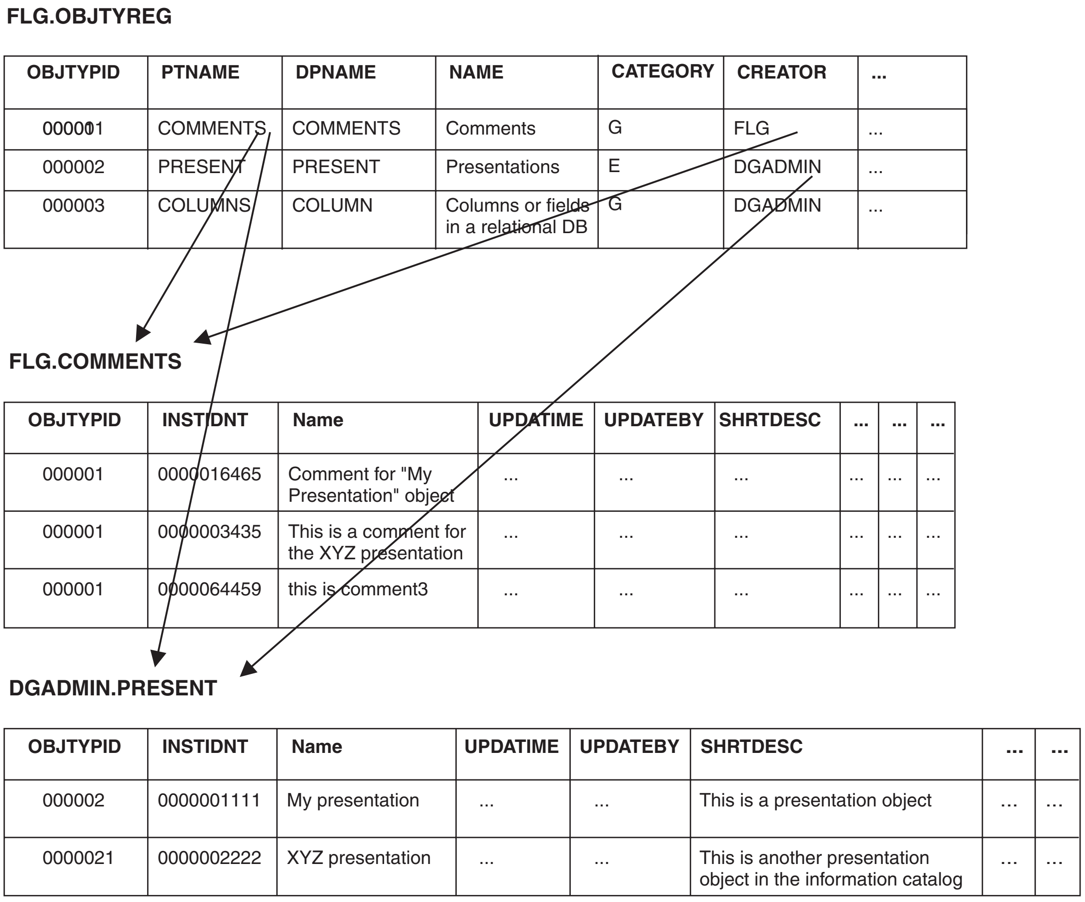


Figure 22. Information Catalog Manager system tables

Notes to Figure 22

1. The relationship between the two tables exists when the values in the OBJTYPID columns of the tables are equal. The relationship is a join between the two tables based on the OBJTYPID column.
2. The relationship between the two tables exists when the values in the OBJTYPID columns of the tables are equal. The relationship is a join between the two tables based on the OBJTYPID column.
3. The relationship between the two tables exists when the values in the DPNAME and HANDLES columns of the tables are equal. The relationship is a join between the two tables based on the DPNAME and HANDLES columns.
4. The relationship between the tables is derived from the PTNAME and CREATOR columns of the FLG.OBJTYREG table, and the physical name of the FLG.COMMENTS table.

For example, in Figure 23, the first entry in the PTNAME column is COMMENTS, and the first entry in the CREATOR column is FLG. Together these values form the fully qualified FLG.COMMENTS table name.



FLG.OBJTYREG

OBJTYPID	PTNAME	DPNAME	NAME	CATEGORY	CREATOR	...
000001	COMMENTS	COMMENTS	Comments	G	FLG	...
000002	PRESENT	PRESENT	Presentations	E	DGADMIN	...
000003	COLUMNS	COLUMN	Columns or fields in a relational DB	G	DGADMIN	...

FLG.COMMENTS

OBJTYPID	INSTIDNT	Name	UPDATIME	UPDATEBY	SHRTDESC	...	...	...
000001	0000016465	Comment for "My Presentation" object	...	...	...	...	...	...
000001	0000003435	This is a comment for the XYZ presentation	...	...	...	...	...	...
000001	0000064459	this is comment3	...	...	...	...	...	...

DGADMIN.PRESENT

OBJTYPID	INSTIDNT	Name	UPDATIME	UPDATEBY	SHRTDESC	...	...
000002	0000001111	My presentation	...	...	This is a presentation object	...	...
0000021	0000002222	XYZ presentation	...	...	This is another presentation object in the information catalog	...	...

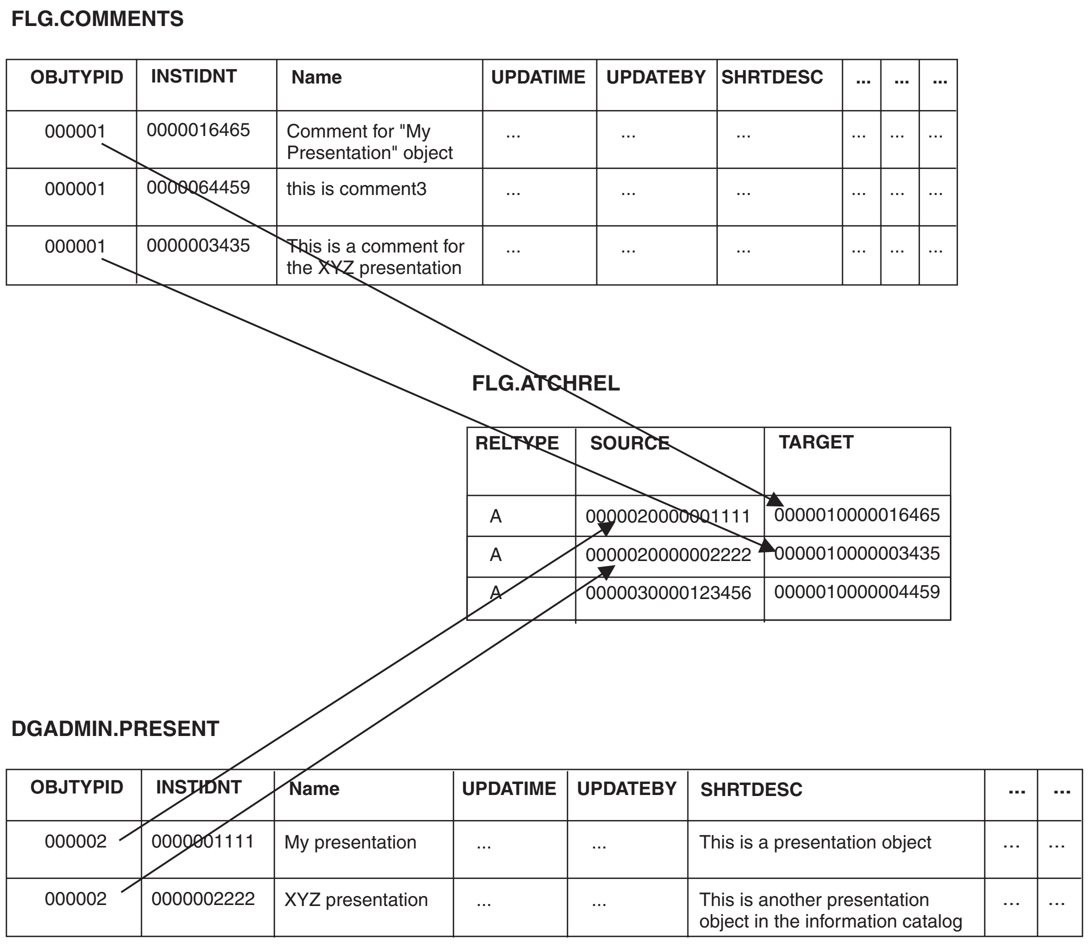
Figure 23. Relationship between table FLG.OBJTYREG and the object type table

5. The relationship between the FLG.OBJTYPREG table and an object type table is derived by concatenating the PTNAME and CREATOR columns of the FLG.OBJTYPREG table. The resulting name is the name of the object type table.

   For example in Figure 23, the second entry in the PTNAME column is PRESENT, and the second entry in the CREATOR column is DGADMIN. Together these values form the fully qualified name DGADMIN.PRESENT.

6. If a relationship is of type A (attaches), the relationship that is stored in the FLG.ATCHREL table is derived by concatenating the object type ID and instance ID of a source table with the object type and instance ID of a target table.

   For example, in Figure 24, the object type and instance ID for DGADMIN.PRESENT are concatenated in the source column of the FLG.ATCHREL table. The concatenated object type and instance ID of the associated comment attached to the presentation object in DGADMIN.PRESENT are stored in the target column.



FLG.COMMENTS

OBJTYPID	INSTIDNT	Name	UPDATIME	UPDATEBY	SHRTDESC	...	...	...
000001	0000016465	Comment for "My Presentation" object	...	...	...	...	...	...
000001	0000064459	this is comment3	...	...	...	...	...	...
000001	0000003435	This is a comment for the XYZ presentation	...	...	...	...	...	...

FLG.ATCHREL

RELTYPE	SOURCE	TARGET
A	0000020000001111	0000010000016465
A	0000020000002222	0000010000003435
A	0000030000123456	0000010000004459

DGADMIN.PRESENT

OBJTYPID	INSTIDNT	Name	UPDATIME	UPDATEBY	SHRTDESC	...	...
000002	0000001111	My presentation	...	...	This is a presentation object	...	...
000002	0000002222	XYZ presentation	...	...	This is another presentation object in the information catalog	...	...

Figure 24. Relationship between FLG.ATCHREL table, source, and target

7. The relationship between each pair of tables is derived from the FLGID of the tables. The FLGID represents the concatenation of the OBJTYPID column and the INSTIDNT column of the tables.

8. The relationship stored in FLG.RELINST is for the following relationships: Contains, Link, and Contact. (See "Logical metadata model" for more information on object category relationships.) The relationship is derived from the FLGID columns of the source table and the target table. See "Predefined Information Catalog Manager object types" on page 151 for more information on Information Catalog Manager object types.
9. The relationship between each pair of tables is derived from the FLGID of the two tables. There might be multiple rows of data in the FLG.OVERDESC table. If so, the rows are sequenced by the SEQNO column of the FLG.OVERDESC table.

Logical metadata model

Every object type must belong to an Information Catalog Manager category. An object type's *category* affects how the Information Catalog Manager handles it. The following list describes the object types that you can create in each of the Information Catalog Manager categories:

Grouping
Object types that can contain other object types.

Elemental
Non-Grouping object types that are the building blocks for other Information Catalog Manager object types.

Contact
Object types that identify a reference for more information about an object. More information might include the name of the person who created the information that the object represents, or the department responsible for maintaining the information.

Program
A Programs object type that identifies and describes applications capable of processing the actual information represented by Information Catalog Manager objects types. The only object type that belongs to the Program category is the Programs object type, which is defined when you create an information catalog.

Dictionary
Object types that define terminology that is specific to your business.

Support
Object types that provide additional information about your information catalog or enterprise.

Attachment
A Comments object type that identifies additional information attached to another Information Catalog Manager object. The only object type that belongs to the Attachment category is the Comments object type, which is defined when you create an information catalog.

Table 75 summarizes the relationships among the Information Catalog Manager object type categories. Figure 25 on page 141 shows a graphical representation of the relationships.

Table 75. Information Catalog Manager category relationships

Category	**Can contain/is contained by**	**Links with**	**Contacts associated**	**Comments attached**	**Programs launch from**
Grouping	Contains other Grouping or Elemental objects	Other Grouping or Elemental objects	Yes	Yes	Yes
Elemental	Contained by any Grouping object	Other Grouping or Elemental objects	Yes	Yes	Yes
Contact	None	None	No	Yes	Yes
Program	None	None	No	Yes	No
Dictionary	None	None	No	Yes	Yes
Support	None	None	No	Yes	Yes
Attachment	None	None	No	No	Yes

You can establish object types for your information catalog in any of three ways:

- Use the object types that come with the Information Catalog Manager in the sample information catalog (see “Predefined Information Catalog Manager object types” on page 151 for information about creating the sample information catalog and a description of the object types that it includes).
- Modify the object types that come with the Information Catalog Manager to fit your organization’s needs (see *Information Catalog Manager Administration Guide* for information about modifying an object type).
- Create your own object types.

Figure 25 on page 141 shows how objects within object type categories are related. In the illustration, parentheses around an object type category name indicate that an object type category is not extendible. Parentheses around an object type name indicate that object type is not extendible. See “Chapter 8. Information Catalog Manager object types” on page 147 for more information on extendible object types.

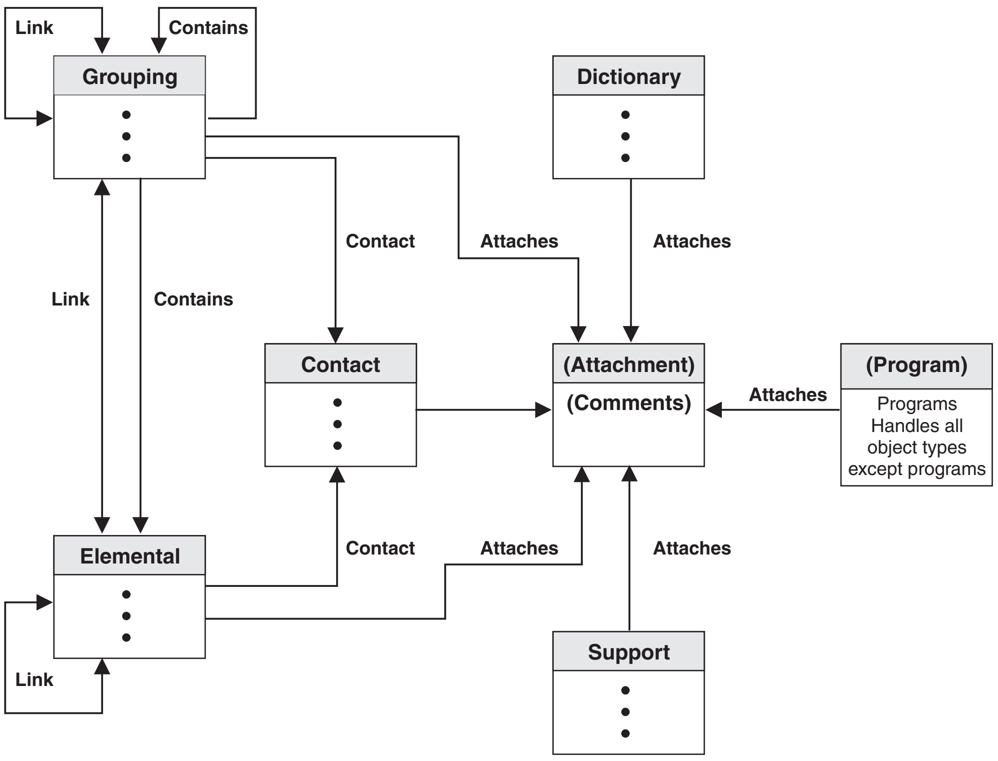


Figure 25. Relationships between object type categories

In Figure 25 above, the following relationships are shown:

Contains
An object can contain many objects, or an object can be contained by many objects.

For example, a Grouping object can contain many Elemental objects, and an Elemental object can be contained by many Grouping objects.

Link An object can be linked to many objects. Objects in a linked relationship are peers, rather than one being an underlying object of the other.

For example, a Grouping object can be linked to many Elemental objects, and an Elemental object can be linked to many Grouping objects.

Contact
An object can have many Contact objects associated with it, or one Contact object can be associated with many objects.

For example, a Grouping object can be associated with many Contact objects, and a Contact object can be associated with many Grouping and Elemental objects.

Attaches
An object can have many Attachment objects associated with it; however, one Attachment object can be associated with only one object.

For example, a Grouping object can have many Attachment objects associated with it; however, one Attachment object can be related to only one Grouping object.

Program
In this relationship, one object type can have many Program object instances associated with it. However, one Program object instance can be associated with only one object type.

For example, an Elemental object type can have many Program object instances associated with it; however, one Program object instance can be associated with only one object type.

Using SQL to access metadata

You can use SQL to extract metadata directly from the database tables that make up the information catalog; this section provides examples.

1. To determine what object type definitions exist in the information catalog, enter the following SQL statement:

   ```
   SELECT OBJTYPID, DPNAME, NAME, CREATOR, PTNAME FROM FLG.OBJTYREG
   ```

   This statement returns the following information:

   OBJTYPID
   Internal identifier for the object type

   DPNAME
   Object type name

   NAME
   External object type name

   CREATOR,PTNAME
   The table (object instance table) where object instances of that type are stored
2. To determine the property names for a specific object type after you determine the object type ID (from step 1), enter the following SQL statement:

```
SELECT  PHYPRPNM, PROPNAME, DATATYPE, LENGTH, OPTIONS, UUISEQNO,
  PROPSEQ FROM FLG.PROPERTY WHERE OBJTYPID = 'object_type_ID'
  ORDER BY PROPSEQ
```

This statement returns the following information (in the order that the properties were created):

PHYPRPNM
Physical column name in the object instance table that maps to an object type property

PROPNAME
Business name of the property

DATATYPE
Data type of the property

LENGTH
Length of the property

OPTIONS
Indicates whether a value is required for this property in the object instance

UUISEQNO
UUI indicator, and sequence number if not 0

PROPSEQ
The order that the properties were added to the properties table

3. To find an instance of a specific object type after you determine the physical tables where the object is stored (from step1 on page 142) and the properties that you want (from step 2 on page 142), enter the following SQL statement:

```
SELECT OBJTYPID, INSTIDNT, NAME,phyprpnm1,phyprpnm2...
  FROM creator.ptname
  WHERE phyprpnm LIKE '%search_criteria%'
```

This statement returns the following information:

OBJTYPID
Internal identifier for the object type

INSTIDNT
Internal identifier for an instance of this object type

phyprpnm1
Value for the property specified in the SELECT statement

phyprpnm2
Value for the property specified in the SELECT statement

In addition, you must enter the following SELECT statement to retrieve any property values that are of the data type long variable character (LONG VARCHAR):

```
SELECT PHYPRPNM, ODESC FROM FLG.OVERDESC
  WHERE OBJTYPID = object_type_ID
  AND INSTIDNT = object_instance_ID
  ORDER BY SEQNO
```

Where `object_type_ID` and `object_instance_ID` are the values that you obtained after you generated the SELECT statement in step 3 on page 143. This statement returns the following information:

PHYPRPNM
: Physical property name of the property that is a long variable character

ODESC
: Value of the long variable character (there might be more than one ODESC for each property value; the order is by sequence)

4. To retrieve a list of all objects in the information catalog, enter the following SQL statement:

```
SELECT FLGID, INSTNAME, TYPENAME FROM FLG.NAMEINST
```

This statement returns the following information:

FLGID
: Concatenated object type and instance IDs for the object

INSTNAME
: External name of the object

TYPENAME
: Type of object (external name for the object type)

5. To determine hierarchical or contact relationships between objects, enter the following statement:

```
SELECT SOURCE, TARGET, RELTYPE FROM FLG.RELINST
```

This statement returns the following information:

SOURCE
: Concatenated object type and instance ID for the object that is the source in a relationship

TARGET
: Concatenated object type and instance ID for the object that is the target of a relationship

RELTYPE
: Relationship type (`C` for container or `T` for contact)

To determine linked or attachment relationships between objects, enter the following SQL statement:

```
SELECT SOURCE, TARGET, RELTYPE FROM FLG.ATCHREL
```

This statement returns the following information:

SOURCE
Concatenated object type and instance ID for the object that is the source in a relationship

TARGET
Concatenated object type and instance ID for the object that is the target of a relationship

RELTYPE
Relationship type (`A` for attachment or `L` for linked)

You can use the SOURCE and TARGET values to look up the object instance information in the object tables. You can also qualify an SQL statement to select specific object values as shown in step 4 on page 146.

Example: You have an application for which you want to display the metadata about a relational table named Employee, and show all of its columns. The object type for Employee is TABLES, and the object type for the columns is COLUMN. Your application includes the following SQL statements:

1. To retrieve the name of the table where TABLES object instances are stored:

   ```
   SELECT OBJTYPID, DPNAME, NAME, CREATOR, PTNAME FROM FLG.OBJTYREG
   WHERE DPNAME = 'TABLES'
   ```

   The statement returns the following information:

   ```
   '000001', 'TABLES', 'Relational Tables', 'USERXYZ', 'TABLES'
   ```
2. To retrieve the OBJTYPID of the COLUMN object:

   ```
   SELECT OBJTYPID, DPNAME, CREATOR, PTNAME from FLG.OBJTYREG
   WHERE DPNAME = 'COLUMN'
   ```

   The statement returns the following information:

   ```
   '000007', 'COLUMN', 'Columns or fields', 'USERXYZ', 'COLUMN'
   ```
3. To retrieve the information about the specific TABLES object for which you want to display metadata:

   ```
   SELECT OBJTYPID, INSTIDNT, NAME, DBNAME, OWNER, TABLES
     FROM USERXYZ.TABLES
     WHERE NAME = 'Employee'
   ```

   The statement returns the following information:

   ```
   '000001', '0040608795',  'Employee', 'MYDBASE', 'USERABC', 'EMPL_TAB'
   ```

4. To retrieve the relationships between the TABLES instance SOURCE and COLUMN instance TARGET:

   ```
   SELECT TARGET FROM FLG.RELINST
   WHERE SOURCE = '0000010040608795'
     AND TARGET LIKE '000007%'
     AND RELTYPE = 'C'
   ```

   The statement returns the following two objects:

   ```
   ('0000079238400354')
   ('0000079843095410')
   ```

5. To retrieve the information about the two returned COLUMN objects:

   ```
   SELECT NAME, SHRTDESC, DATATYPE, LENGTH FROM USERXYZ.COLUMNS
   WHERE INSTIDNT IN ('9238400354', 9843095410')
   ```

   The statement returns the following information:

   ```
   ('Name', 'Employee name information', 'CHAR', '80')
   ('Address', 'Employee address information', 'CHAR', '220')
   ```

Chapter 8. Information Catalog Manager object types

This chapter provides detailed information about Information Catalog Manager object types.

Default properties for all object types

The Information Catalog Manager provides a set of default properties for the generic object type. These default properties serve as the base for any user-defined tables. Some properties are generated by the Information Catalog Manager; some are required; and some are optional.

FLGID
An ID, generated by the Information Catalog Manager, that uniquely identifies an instance.

The FLGID ID is 16 digits, with the first 6 digits used for the object type ID (OBJTYPID) and the next 10 digits used for the instance ID (INSTIDNT). FLGID has the following format:

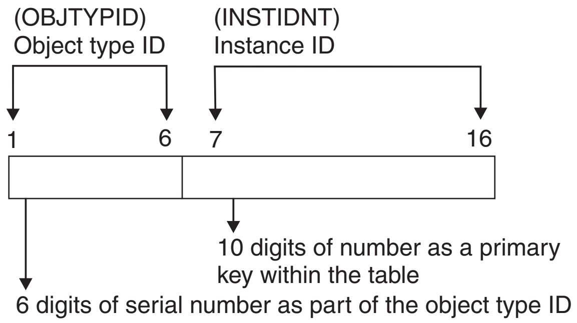

(OBJTYPID)
Object type ID
(INSTIDNT)
Instance ID
1
6
7
16
10 digits of number as a primary key within the table
6 digits of serial number as part of the object type ID

Figure 26. FLGID Format

Name Name of the step. The name can be used on glossary, news queries, and other objects. This is a required property, and it is not nullable. It is displayed in the Information Catalog Manager windows.

UPDATIME
A system time stamp that indicates the date and time of the creation or last update to the instance.

UPDATEBY
The user ID of the information catalog administrator or user with special privileges who last updated the instance. For Attachment objects, this field can be the user ID of an information catalog user.

Default properties for all object types

Default property summary

The information catalog administrator can use the predefined template to create an object type. The information catalog administrator can append attributes to the template to customize it for the organization. The predefined template has several optional fields. Table 76 shows the default properties.

Table 76. Default properties of the predefined template

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	A six-digit object type ID, generated by the Information Catalog Manager, that represents a specific object type.	No	SBCS
INSTIDNT	CHAR(10)	The unique instance ID generated by the Information Catalog Manager. It is the second part of the FLGID, the 10-digit serial number that uniquely identifies this instance within its own object type.	No	SBCS
NAME	VARCHAR(80)	This name is entered by the information catalog user to identify each user-defined object instance in the product.	No	Both SBCS and DBCS
UPDATIME	CHAR (26)	The date and time of metadata creation or last update. This value is generated by the Information Catalog Manager.	No	SBCS
UPDATEBY	CHAR(8)	The user ID of the information catalog administrator or user with special update privileges who last updated the instance. For attachment objects this field might be the user ID of the information catalog user. This value is generated by the Information Catalog Manager.	No	Both SBCS and DBCS
Note: NLS: National Language Support SBCS: Single Byte Character Set DBCS: Double Byte Character Set				

Guidelines for extendible objects types

1. An object type is extendible if it can be changed. An object type category is extendible if other objects can be added to it. Most Information Catalog Manager objects are extendible including PROGRAMS, QUERY, IMAGE, REPORT, business group (BUSNSGP), TABLES, COLUMNS, GLOSSARY, CONTACTS, and NEWS. The COMMENTS object type is not extendible. The Programs and Attachments categories are not extendible.

2. All Information Catalog Manager objects are organized into the following categories:

   Elemental (E)
   An object type that cannot have any objects within it, for example, REPORT, QUERY, and IMAGE objects.

   Grouping (G)
   An object type that can contain other Grouping or Elemental objects, for example, INFOGRPS, and TABLES object types.

   Program (P)
   An executable object type, for example, the PROGRAMS object type.

   Contact (C)
   A special object type used to identify a person or organization to contact if a question arises about another object, for example, the CONTACTS object type.

   Dictionary (D)
   An object type that helps the user find the definition or synonyms of the terminology used in the user's business environment, for example, the GLOSSARY object type.

   Support (S)
   An object type that provides additional information about the information catalog or business environment, for example, the NEWS object type.

   Attachment (A)
   An object type that is used to attach additional information to another object, for example, the COMMENTS object type.

   The process used to create, delete, and update object types is identical for all object types, except for the PROGRAMS and COMMENTS object types.

   The PROGRAMS object type is predefined by the Information Catalog Manager and is the only object type used within the Program category. You cannot create another object type under the Program category, and you cannot delete the PROGRAMS object type.

   The COMMENTS object type is predefined by the Information Catalog Manager and is the only object type used within the Attachment category. You cannot create another object type under the Attachment category, and you cannot delete the COMMENTS object type.

3. With a new object type such as VIDEO or AUDIO, you can create your own object type, if the DPname of the object type is unique within the Information Catalog Manager.

4. All objects must include a universal unique identifier, UUI, as part of their object type definition. The UUI is used to compare with a similar identifier in the target information catalog during the import process.
5. If the property has a data type such as LONG VARCHAR, the Information Catalog Manager will automatically put the property and its metadata into a separate overflow table and split the property into smaller segments so a user can search for it. The search will proceed slowly because of the size of the property.
6. The Information Catalog Manager supports five data types:

   CHAR
   A fixed character string, up to 254 characters.

   VARCHAR
   A variable-length character string, up to 4000 characters. The maximum length of a row of a table is also 4000.

   LONG VARCHAR
   A variable-length character string, up to 32700 characters.

   The Information Catalog Manager keeps metadata of this type in a separate table and divides the metadata into smaller segments so that you can search for the string. When the metatdata is retrieved, the Information Catalog Manager puts the segments back together.

   TIMESTAMP
   A seven-part value that consists of year, month, day, hour, minute, second, and microsecond in a character string of 26 bytes. It has the format yyyy-mm-dd-hh.mm.ss.nnnnnn.

   LONG VARCHAR FOR BIT DATA
   Binary data such as a bitmap.

Relation types

1. The Information Catalog Manager supports the following types of relationships that are created and deleted through the same FLGRelation API. Different APIs, such as FLGNavigate, FLGWhereUsed, and FLGListContacts are used to access each type of the relationship. These APIs call their corresponding IPIs to complete the user's request.
   a. Contains (C)

      For example: a hierarchical business structure or a relational table to the relational columns.

      This relation is retrieved by APIs such as FLGNavigate and FLGWhereUsed.
   b. Contact (T)

      For example: the name of a person providing services for specified objects.

The FLGListContacts API is used to access this relation.

c. Attaches relationship (A)

   For example: comments for a specified object.

   The FLGListAssociates and FLGFoundIn API are used to retrieve this relation.

d. Link relationship (L)

   A grouping or elemental category object type instance can link to any other grouping or elemental category object type instance.

   The FLGListAssociates API is used to retrieve this relation.

2. The relation rules based on the Information Catalog Manager defined categories are described in “Logical metadata model” on page 139.

Objects are not required to have relationships. You can find all objects by using the Information Catalog Manager windows (see the *Information Catalog Manager Administration Guide*), the FLGSearch API, or by viewing the FLG.NAMEINST table. See the *Information Catalog Manager Programming Guide and Reference* for more information on Information Catalog Manager APIs. See “FLG.NAMEINST table” on page 124 for information on the FLG.NAMEINST table.

Relation instance

If there is a relation between two object instances, this instance-to-instance relation is added to the relation instance table.

The table has the following format:

FLGID of source (16 digits)	FLGID of target (16 digits)	RelType C/T/L/A

See “FLG.RELINST table” on page 132 for more information on the properties in the table.

Predefined Information Catalog Manager object types

The Information Catalog Manager includes predefined object types that can be exchanged with metadata from other Data Warehouse Center components and other MDIS-conforming products from IBM and other companies. This section describes all of the predefined Information Catalog Manager object types, including how the object type properties map to MDIS object types. For information about the Metadata Interchange Specification, including complete MDIS object type definitions, go to the Meta Data Coalition’s Web site at http://www.MDCinfo.com.

Predefined Information Catalog Manager object types

The Information Catalog Manager provides both the predefined object types and sample objects of each type within the sample information catalog. The sample information catalog includes at least one object type for each of the seven Information Catalog Manager categories. This section describes how to create the sample information catalog. For details of Information Catalog Manager object type capabilities, see the *Information Catalog Manager Administration Guide*.

Table 77 lists all the object types in the sample information catalog. Object types can represent data or a relationship between two object types.

Object types that represent data
Most predefined object types represent types of data such as the Charts or Documents object types.

Object types that represent relationships
The Transformations object type is a special object type that represents a relationship between two other object types. Specifically, it represents the transformation of data from the data's source format to its target format. You can use Transformations object types to provide information about the lineage of the data within a target relational database.

Table 77. Predefined data object types summary

Object type name	Description	Properties defined on page
Application data	Internal use only	160
Audio clips	Represents files that contain audio information	193
Business subject areas	Represents logical grouping of objects	162
Charts	Represents either printed or electronic charts	194
Columns or fields	Represents columns within a relational table, fields within a file, or fields within an IMS segment	163
Comments	Contains comments about other objects in the information catalog	211
Databases	Represents relational databases	166
Information Catalog Manager news	Conveys information about changes to the information catalog	207
Dimensions within a multi-dimensional database	Represents dimensions within a multidimensional database	167

Table 77. Predefined data object types summary (continued)

Object type name	Description	Properties defined on page
Documents	Represents books, manuals, and technical papers	195
Elements	Represents MDIS Element objects that do not map directly to the “Columns or fields” object type	170
Files	Represents a file within a file system	172
Glossary entries	Represents definitions for terms used in the information catalog	205
Images or graphics	Represents graphic images, such as bitmaps	196
IMS database definitions (DBD)	Represents IMS database definitions	174
IMS program control blocks (PCB)	Represents IMS program control blocks	176
IMS program specification blocks (PSB)	Represents IMS program specification blocks	177
IMS segments	Represents IMS segments	178
Internet documents	Represents Web sites and other documents on the Internet that might be of interest	198
Lotus® Approach® queries	Represents available Lotus Approach queries for use with your organization’s data	199
Members within a multidimensional database	Represents a member within a multidimensional database	180
Multidimensional databases	Represents multidimensional databases	183
Online news services	Represents news and information services that can be accessed online	208
Online publications	Represents publications and other documents that can be accessed from online services	209
People to contact	Identifies a person or group that is responsible for single or multiple objects within the information catalog	204

Table 77. Predefined data object types summary (continued)

Object type name	Description	Properties defined on page
Presentations	Represents printed or electronic presentations	200
Programs that can be invoked from Information Catalog Manager objects	Defines an application capable of processing a particular object type	210
Records	Represents MDIS Record objects that do not map directly to the “Files” or “Relational tables or views” object type	184
Relational tables and views	Represents tables or views of relational databases	186
Subschemas	Represents logical groupings of records within a database	189
Transformations	Represents expressions or logic used to populate columns of data within the target relational database	191
Spreadsheets	Represents desktop spreadsheets (for example, Lotus 1-2-3® or Microsoft Excel spreadsheets)	201
Text-based reports	Represents either printed or electronic reports	202
Video clips	Represents files that contain video information	203

Predefined object type models

The Information Catalog Manager predefined object types participate in the six data models shown in Figures 27 through 32.

Figure 27 shows the object types that participate in the relational model.

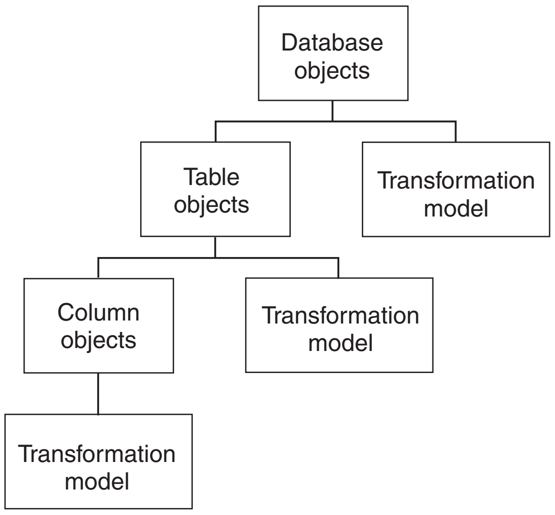


Figure 27. Relational model and the predefined object types

Figure 28 shows the object types that participate in the hierarchical models.

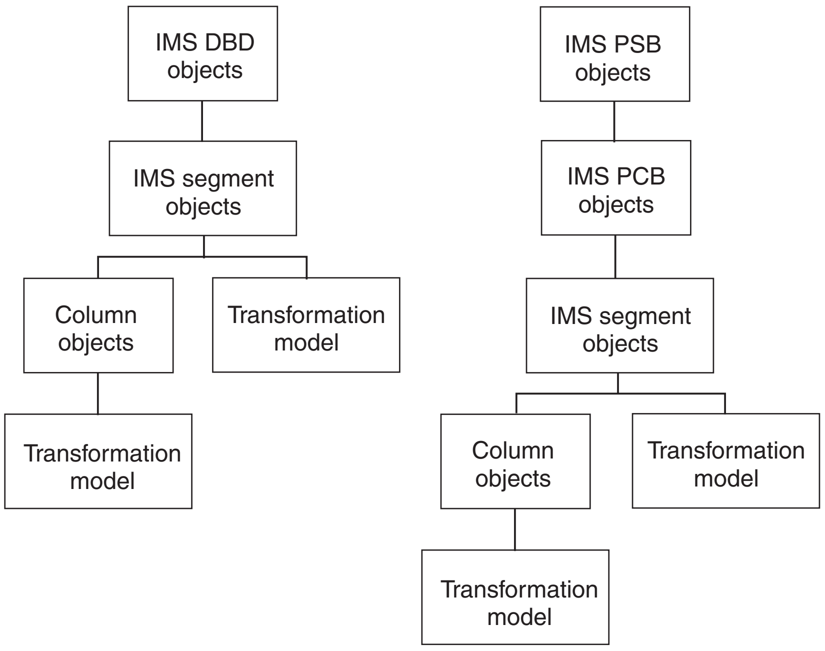


Figure 28. Hierarchical models and the predefined object types

Figure 29 shows the object types that participate in the file models.

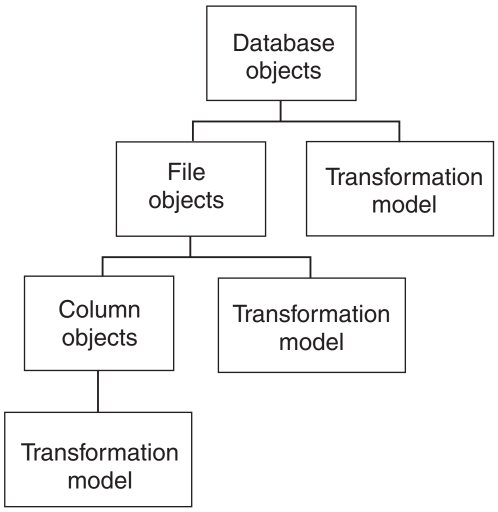


Figure 29. File models and the predefined object types

Figure 30 shows the object types that participate in the multi-dimensional (OLAP) model.

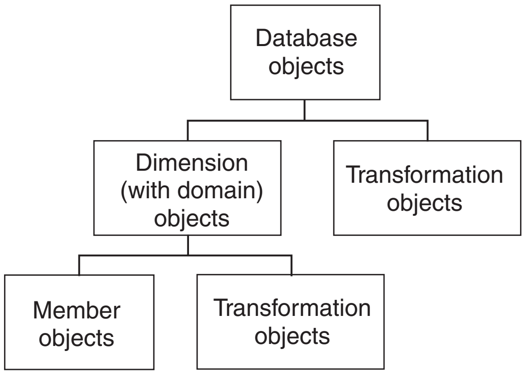


Figure 30. Multi–dimensional model and the predefined object types

Figure 31 shows the object types that participates in the transformation models.

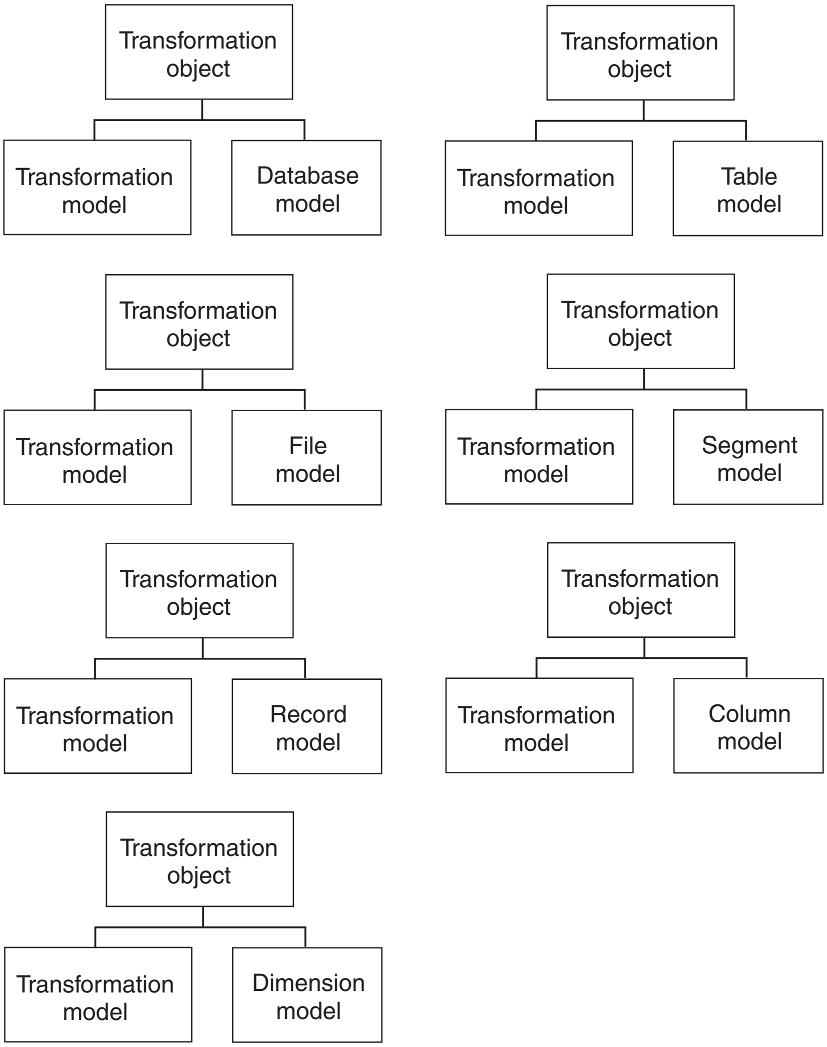


Figure 31. Transformation model and the predefined object types

Figure 32 shows the object types that participates in the subject area model.

Predefined object types in the sample information catalog

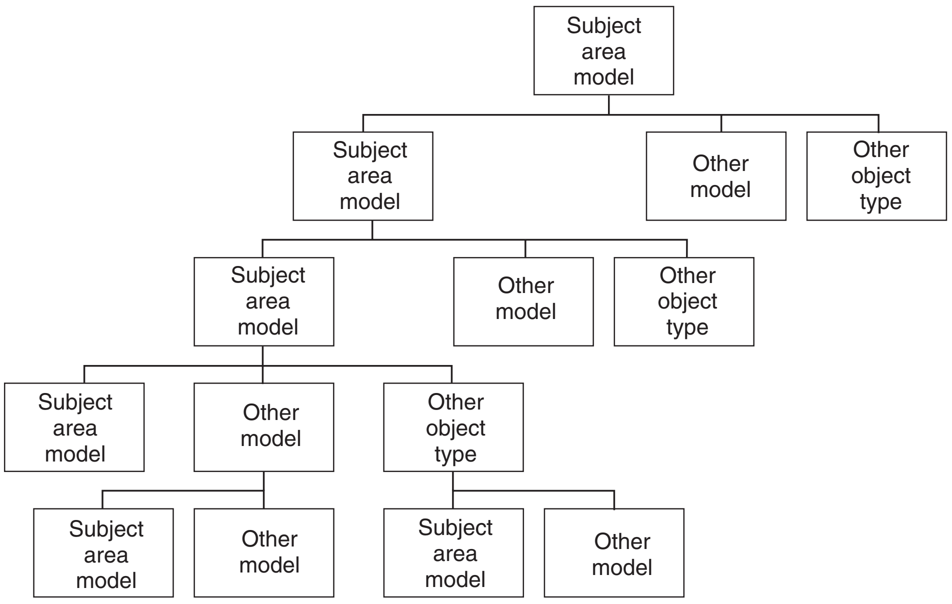


Figure 32. Subject area model and the predefined object types

Predefined object type descriptions

Sample Information Catalog Manager object types are organized by category and defined in the tables that start on page 160.

Each table lists the properties for that object type. Each property is described in terms of its property specifications. A *property's specifications* govern the value that you can give that property when creating or updating an object of that object type.

The property specifications are:

EXTNAME	The name of the property; for example, Business Name.
DT	The data type of the property's value; for example, CHAR or VARCHAR.
DL	The length (maximum length for VARCHAR or LONG VARCHAR data types) of the value for the property.
SHRTNAME	The name used to identify the property within the Information Catalog Manager data store.

NULLS	**R**	A value for the property is required; value for NULLS in the tag language is N.
	O	A value for the property is optional; value for NULLS in the tag language is Y.
	S	A value generated by the Information Catalog Manager that indicates that provides a value for the property when any object is created. You cannot specify this value.
UUISEQ	If the property is part of the UUI, then this number indicates its position within the UUI.	

MDIS mappings

Tables that describe Information Catalog Manager object type properties begin on page 160. For each object type that conforms to the Metadata Interchange Specification (MDIS), the MDIS equivalent for each property appears in the column titled **Maps to MDIS name.**

1. Find the table for the object type you are exporting.
2. Find the MDIS name in the **Maps to MDIS name** column.
3. Find the equivalent Information Catalog Manager names in the **Property name** and **Property short name** columns.

Each property described in the following object type property tables corresponds to a column with the same property short name in the Information Catalog Manager DB2 storage table XXX.*object_type_name*, where *object_type_name* is the name of the object type described in the table. If the property data type is LONG VARCHAR, the property corresponds to a row in the Information Catalog Manager DB2 storage table FLG.OVERDESC.

Grouping category

The Grouping category contains the following object types:

- "Application data" on page 160
- "Business subject areas" on page 161
- "Columns or fields" on page 162
- "Databases" on page 165
- "Dimensions within a multidimensional database" on page 167
- "DWC Process" on page 169
- "Elements" on page 170
- "Files" on page 171

- "IMS database definitions (DBD)" on page 173
- "IMS program control blocks (PCB)" on page 175
- "IMS program specification blocks (PSB)" on page 177
- "IMS segments" on page 178
- "Members within a multidimensional database" on page 180
- "Multidimensional databases" on page 182
- "Records" on page 184
- "Relational tables and views" on page 186
- "Star Schemas" on page 188
- "Subschemas" on page 189
- "Transformations" on page 190

Application data: Used by the Information Catalog Manager for some MDIS metadata exchanges. Objects of this object type might appear in your information catalog, but you do notuse this object type to create objects.

The tag language for defining this object type is in the file FLGNYAPL.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.APPLDATA. For more information on table names, see "MDIS mappings" on page 159.

Table 78 provides information about the properties of the Application data object type.

Table 78. Properties of the Application data object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	O	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	

Table 78. Properties of the Application data object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Long description	LONG VARCHAR	32700	LONGDESC	O	
Source object identifier	CHAR	16	FLGID	R	1
Application data field 0	LONG VARCHAR	32700	APPLDAT0	O	
Application data field 1	LONG VARCHAR	32700	APPLDAT1	O	
Application data field 2	LONG VARCHAR	32700	APPLDAT2	O	
Application data field 3	LONG VARCHAR	32700	APPLDAT3	O	
Application data field 4	LONG VARCHAR	32700	APPLDAT4	O	
Application data field 5	LONG VARCHAR	32700	APPLDAT5	O	
Application data field 6	LONG VARCHAR	32700	APPLDAT6	O	
Application data field 7	LONG VARCHAR	32700	APPLDAT7	O	
Application data field 8	LONG VARCHAR	32700	APPLDAT8	O	
Application data field 9	LONG VARCHAR	32700	APPLDAT9	O	
Timestamp source definition created	CHAR	26	CRTTIME	O	
Timestamp source definition last changed	CHAR	26	SRCDATCF	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Business subject areas: Represents logical groupings of objects.

The tag language for defining this object type is in the file FLGNYINF.TYP in the \VWSWIN\DGWIN\TYPES directory.

Business subject areas

The Information Catalog Manager DB2 storage table name for this object type is XXX.INFOGRPS. For more information on table names, see “MDIS mappings” on page 159.

Table 79 provides information about the properties of the Business subject areas object type.

Table 79. Properties of the Business subject areas object type

Property name	**Data type**	**Size**	**Property short name**	**Value flag**	**UUI order**
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	1
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Filename	VARCHAR	254	FILENAME	O	
URL to access data	VARCHAR	254	URL	O	
For more information . . .	VARCHAR	80	CONTACT	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Columns or fields: Represents columns within a relational table, fields within a file, or fields within an IMS segment.

The tag language for defining this object type is in the file FLGNYCOL.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.COLUMNS. For more information on table names, see "MDIS mappings" on page 159.

Table 80 provides information about the properties of the Columns or fields object type.

Table 80. Properties of the Columns or fields object type. The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		ElementLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
Catalog remarks	VARCHAR	254	REMARKS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Column or field last refreshed	CHAR	26	FRESHDAT	O		ElementLastRefreshDate
Data type of column or field	CHAR	30	DATATYPE	O		ElementDataType
Length of column or field	CHAR	20	LENGTH	O		ElementLength
Scale of column or field	CHAR	5	SCALE	O		ApplicationData

Columns or fields

Table 80. Properties of the Columns or fields object type (continued). The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name
Precision of column or field	CHAR	5	PRECDIG	O		ElementPrecision
Can column or field be null	CHAR	1	NULLS	O		ElementNulls
Column or field ordinality	CHAR	5	ORDINAL	O		ElementOrdinality
Column or field position	CHAR	5	POSNO	O		ElementPosition
Byte offset of column or field from start	CHAR	10	STARTPOS	O		ApplicationData
Is column or field part of a key	CHAR	1	ISKEY	O		ApplicationData
Is column or field a unique key	CHAR	1	UNIQKEY	O		ApplicationData
Position of column or field within key	CHAR	5	KEYPOSNO	O		ElementKeyPosition
Database host server name	VARCHAR	80	SERVER	O		ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	1	DatabaseName
Table owner	VARCHAR	80	OWNER	R	2	OwnerName
Table name	VARCHAR	80	TABLES	R	3	RecordName
Column or field name	VARCHAR	254	COLUMNS	R	4	ElementName
Filename	VARCHAR	254	FILENAME	R	5	ApplicationData
URL to access data	VARCHAR	254	URL	O		ApplicationData
Containing dimension	VARCHAR	80	DIMENSION	O		DimensionName

Table 80. Properties of the Columns or fields object type (continued). The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name
Is data a before or after image, or computed	CHAR	50	COLIMAGE	O		ApplicationData
Source column or field name or expression used to populate column	VARCHAR	254	COLEXPR	O		ApplicationData
String used to represent null values	VARCHAR	30	IDSNREP	O		ApplicationData
Resolution of dates	CHAR	1	IDSRES	O		ApplicationData
Is data text	CHAR	1	ISTEXT	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Databases: Represents relational databases.

The tag language for defining this object type is in the file FLGNYDAT.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.DATABAS. For more information on table names, see “MDIS mappings” on page 159.

Table 81 on page 166 provides information about the properties of the Databases object type.

Databases

Table 81. Properties of the Databases object type. The MDIS name for this object type is Database.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		DatabaseLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database owner	VARCHAR	80	OWNER	O		OwnerName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database server type	VARCHAR	80	SRVRTYPE	O		ServerType
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
Database type	VARCHAR	80	DBTYPE	R	3	DatabaseType
Database extended type	VARCHAR	40	DBETYPE	O		DatabaseExtendedType
Database status	VARCHAR	80	DBSTAT	O		DatabaseStatus
Database location	VARCHAR	80	LOCATION	O		ApplicationData
URL to access data	VARCHAR	254	URL	O		ApplicationData
System code page	VARCHAR	10	CODEPAGE	O		ApplicationData

Table 81. Properties of the Databases object type (continued). The MDIS name for this object type is Database.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Agent type	VARCHAR	80	AGENTTYPE	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by the Information Catalog Manager, R = required, O = optional						

Dimensions within a multidimensional database: Represents dimensions within a multidimensional database. A dimension is comprised of members.

The tag language for defining this object type is in the file FLGNYDIM.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.DIMENSION. For more information on table names, see “MDIS mappings” on page 159.

Table 82 provides information about the properties of the dimensions within a multidimensional database object type.

Table 82. Properties of the Dimensions within a multidimensional database object type. The MDIS name for this object type is Dimension.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		DimensionLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		

Dimensions within a multidimensional database

Table 82. Properties of the Dimensions within a multidimensional database object type (continued). The MDIS name for this object type is Dimension.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database last refreshed	CHAR	26	FRESHDAT	O		ApplicationData
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
Using application name	VARCHAR	80	APPLNAME	R	3	ApplicationData
Dimension owner	VARCHAR	80	OWNER	O		OwnerName
Dimension name	VARCHAR	80	DIMENSON	R	4	DimensionName
Dimension class or type	VARCHAR	80	TYPE	O		DimensionType
Total member count	CHAR	10	TOTALCNT	O		DimensionCount
Level count	CHAR	10	LEVELCNT	O		DimensionLevelCount
Application-specific information	VARCHAR	512	APPLDATA	O		ApplicationData
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated

Table 82. Properties of the Dimensions within a multidimensional database object type (continued). The MDIS name for this object type is Dimension.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by the Information Catalog Manager, R = required, O = optional						

DWC Process: Represents a process in the Data Warehouse Center.

The tag language for defining this object type is in the file FLGNYINF.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.DWCPROC. For more information on table names, see “MDIS mappings” on page 159.

Table 83 provides information about the properties of the Business subject areas object type.

Table 83. Properties of the DWC Process object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Name	VARCHAR	80	NAME	R	1
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
For further information . . .	VARCHAR	80	RESPNSBL	O	
URL to access data	VARCHAR	254	URL	O	
Timestamp source definition last changed	CHAR	26	SRCDATCF	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Elements: Represents MDIS element objects that do not map directly to the Columns or fields object type.

The tag language for defining this object type is in the file FLGNYELE.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.ELEMENT. For more information on table names, see "MDIS mappings" on page 159.

Table 84 provides information about the properties of the Elements object type.

Table 84. Properties of the Elements object type. The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
For further information...	VARCHAR	80	RESPNSBL	O	
Element last refreshed	CHAR	26	FRESHDAT	O	
Database host server name	VARCHAR	80	SERVER	R	1
Database or subsystem name	VARCHAR	80	DBNAME	R	2

Table 84. Properties of the Elements object type (continued). The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order
Element owner	VARCHAR	80	OWNER	R	3
Dimension or record name	VARCHAR	80	DIMRECNM	R	4
Element name	VARCHAR	80	ELEMNAME	R	5
URL to access data	VARCHAR	254	URL	O	
Data type of element	CHAR	30	DATATYPE	O	
Length of element	CHAR	20	LENGTH	O	
Scale of element	CHAR	5	SCALE	O	
Precision of element	CHAR	5	PRECDIG	O	
Can element be null	CHAR	1	NULLS	O	
Position of element within primary key	CHAR	5	KEYPOSNO	O	
Element position	CHAR	5	POSNO	O	
Element ordinality	CHAR	5	ORDINAL	O	
Timestamp source definition created	CHAR	26	CRTTIME	O	
Timestamp source definition last changed	CHAR	26	SRCDATCF	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Files: Represents a file within a file system.

Files

The tag language for defining this object type is in the file FLGNYFIL.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.FILE. For more information on table names, see "MDIS mappings" on page 159.

Table 85 provides information about the properties of the Files object type.

Table 85. Properties of the Files object type. The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		RecordLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short Description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long Description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
File owner	VARCHAR	80	OWNER	R	3	OwnerName
File path or directory	VARCHAR	254	FILEPATH	R	4	ApplicationData
File filename	VARCHAR	254	FILENAME	R	5	RecordName
File data last refreshed	CHAR	26	FRESHDAT	O		RecordLastRefreshDate

Table 85. Properties of the Files object type (continued). The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Transformation program last run	CHAR	26	LASTRUN	O		ApplicationData
Transformation program run frequency	VARCHAR	80	RUNFREQ	O		RecordUpdateFrequency
Transformation program type	VARCHAR	32	SOURCE	O		ApplicationData
Partial or full file copy/update	CHAR	1	COPYCOMP	O		ApplicationData
Copied/updated data is in a consistent state	CHAR	1	CONSIST	O		ApplicationData
Transformation program last changed	CHAR	26	PGMGEND	O		ApplicationData
Transformation program last compiled	CHAR	26	PGMCOMP	O		ApplicationData
File class or type	VARCHAR	80	TYPE	O		RecordType
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated

Note: S = generated by the Information Catalog Manager, R = required, O = optional

IMS database definitions (DBD): Represents IMS database definitions.

The tag language for defining this object type is in the file FLGNYDBD.TYP in the \VWSWIN\DGWIN\TYPES directory.

IMS database definitions (DBD)

The Information Catalog Manager DB2 storage table name for this object type is XXX.IMSDBD. For more information on table names, see “MDIS mappings” on page 159.

Table 86 provides information about the properties of the IMD database definitions (DBD) object type.

Table 86. Properties of the IMS database definitions (DBD) object type. The MDIS name for this object type is Database.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		DatabaseLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
Database last refreshed	CHAR	26	FRESHDAT	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database owner	VARCHAR	80	OWNER	O		OwnerName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database server type	VARCHAR	80	SRVRTYPE	O		ServerType
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName

Table 86. Properties of the IMS database definitions (DBD) object type (continued). The MDIS name for this object type is Database.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Database type	VARCHAR	80	DBTYPE	R	3	DatabaseType
Database extended type	VARCHAR	40	DBETYPE	O		ApplicationData
Database status	VARCHAR	80	DBSTAT	O		DatabaseStatus
IMS access method	VARCHAR	80	IMSACC	O		ApplicationData
Operating system access method	VARCHAR	80	OSACC	O		ApplicationData
Shared index names	VARCHAR	320	SHRINDEX	O		ApplicationData
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated

Note: S = generated by the Information Catalog Manager, R = required, O = optional

IMS program control blocks (PCB): Represents IMS program control blocks.

The tag language for defining this object type is in the file FLGNYPCB.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.IMSPCB. For more information on table names, see “MDIS mappings” on page 159.

Table 87 on page 176 provides information about the properties of the IMS program control blocks (PCB) object type.

IMS program control blocks (PCB)

Table 87. Properties of the IMS program control blocks (PCB) object type. The MDIS name for this object type is Subschema.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		SubschemaLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
PCB name	VARCHAR	80	PCBNAME	R	3	SubschemaName
PCB owner	VARCHAR	80	OWNER	O		OwnerName
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated

Note: S = generated by the Information Catalog Manager, R = required, O = optional

IMS program specification blocks (PSB): Represents IMS program specification blocks.

The tag language for defining this object type is in the file FLGNYPSB.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.PSB. For more information on table names, see "MDIS mappings" on page 159.

Table 87 on page 176 provides information about the properties of the IMS program specification blocks (PSB) object type.

Table 88. Properties of the IMS program specification blocks (PSB) object type. The MDIS name for this object type is Subschema.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		DatabaseLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database server type	VARCHAR	80	SRVRTYPE	O		ServerType
Database type	VARCHAR	80	DBTYPE	R	3	DatabaseType
Database extended type	VARCHAR	40	DBETYPE	O		ApplicationData
Database status	VARCHAR	80	DBSTAT	O		DatabaseStatus

Table 88. Properties of the IMS program specification blocks (PSB) object type (continued). The MDIS name for this object type is Subschema.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
PSB name	VARCHAR	80	PSBNAME	R	2	DatabaseName
PSB owner	VARCHAR	80	OWNER	O		OwnerName
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by the Information Catalog Manager, R = required, O = optional						

IMS segments: Represents IMS segments.

The tag language for defining this object type is in the file FLGNYSEG.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.IMSSEG. For more information on table names, see "MDIS mappings" on page 159.

Table 89 provides information about the properties of the IMS segments object type.

Table 89. Properties of the IMS segments object type. The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		RecordLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		

Table 89. Properties of the IMS segments object type (continued). The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Short Description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long Description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
Segment last refreshed	CHAR	26	FRESHDAT	O		RecordLastRefreshDate
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database host server name	VARCHAR	80	SERVER	O		ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	1	DatabaseName
Segment name	VARCHAR	80	SEGNAME	R	2	RecordName
Segment owner	VARCHAR	80	OWNER	O		OwnerName
Segment type	VARCHAR	80	TYPE	O		RecordType
Segment maximum length	CHAR	5	MAXLEN	O		ApplicationData
Segment minimum length	CHAR	5	MINLEN	O		ApplicationData
Real logical child segment source	CHAR	20	PSEGSRC	O		ApplicationData
Logical parent concatenated key source	CHAR	20	LPCKSRC	O		ApplicationData
Transformation program last run	CHAR	26	LASTRUN	O		ApplicationData
Transformation program run frequency	VARCHAR	80	RUNFREQ	O		RecordUpdateFrequency

Table 89. Properties of the IMS segments object type (continued). The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Members within a multidimensional database: Represents a member within a multidimensional database. A member is part of a dimension, and a dimension is part of a multidimensional database.

The tag language for defining this object type is in the file FLGNYMEM.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.MEMBER. For more information on table names, see "MDIS mappings" on page 159.

Table 90 provides information about the properties of the Members within a multidimensional database object type.

Table 90. Properties of the Members within a multidimensional database object type. The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	CHAR	80	NAME	R		ElementLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		

Table 90. Properties of the Members within a multidimensional database object type (continued). The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information	VARCHAR	80	RESPNSBL	O		ContactName
Member last refreshed	CHAR	26	FRESHDAT	O		ElementLastRefreshDate
Member owner	VARCHAR	80	OWNER	O		OwnerName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
Using application name	VARCHAR	80	APPLNAME	R	3	ApplicationData
Dimension name	VARCHAR	80	DIMENSON	R	4	DimensionName
Member name	VARCHAR	80	MEMBER	R	5	ElementName
Data type of member	CHAR	30	DATATYPE	O		ElementDataType
Length of member	CHAR	20	LENGTH	O		ElementLength
Scale of member	CHAR	5	SCALE	O		ApplicationData
Precision of member	CHAR	5	PRECDIG	O		ElementPrecision
Can member be null	CHAR	1	NULLS	O		ElementNulls

Table 90. Properties of the Members within a multidimensional database object type (continued). The MDIS name for this object type is Element.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Position of member within primary key	CHAR	5	KEYPOSNO	O		ElementKeyPosition
Member position	CHAR	5	POSNO	O		ElementPosition
Member ordinality	CHAR	5	ORDINAL	O		ElementOrdinality
Derived from...	VARCHAR	512	DERIVED	O		ApplicationData
Application-specific information	VARCHAR	512	APPLDATA	O		ApplicationData
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by the Information Catalog Manager, R = required, O = optional						

Multidimensional databases: Represents multidimensional databases.

The tag language for defining this object type is in the file FLGNYOLA.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.OLAPMODL. For more information on table names, see “MDIS mappings” on page 159.

Table 91 on page 183 provides information about the properties of the Multidimensional databases object type.

Table 91. Properties of the Multidimensional databases object type. The MDIS name for this object type is Database.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		DatabaseLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database last refreshed	CHAR	26	FRESHDAT	O		ApplicationData
Database owner	VARCHAR	80	OWNER	O		OwnerName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database server type	VARCHAR	80	SRVRTYPE	O		ServerType
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
Database type	VARCHAR	80	DBTYPE	O		DatabaseType
Database extended type	VARCHAR	20	DBETYPE	O		ApplicationData
Database status	VARCHAR	80	DBSTAT	O		DatabaseStatus
Using application name	VARCHAR	80	APPLNAME	R	3	ApplicationData

Table 91. Properties of the Multidimensional databases object type (continued). The MDIS name for this object type is Database.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Application-specific information	VARCHAR	512	APPLDATA	O		ApplicationData
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by the Information Catalog Manager, R = required, O = optional						

Records: Represents MDIS Record objects that do not map directly to the "Files" or "Relational tables or views" object types. Records are comprised of elements.

The tag language for defining this object type is in the file FLGNYREC.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.RECORD. For more information on table names, see "MDIS mappings" on page 159.

Table 92 provides information about the properties of the Records object type.

Table 92. Properties of the Records object type. The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		RecordLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		

Table 92. Properties of the Records object type (continued). The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
Record owner	VARCHAR	80	OWNER	R	3	OwnerName
Record name	VARCHAR	80	RECNAME	R	4	RecordName
Record data last refreshed	CHAR	26	FRESHDAT	O		RecordLastRefreshDate
Transformation program last run	CHAR	26	LASTRUN	O		ApplicationData
Transformation program run frequency	VARCHAR	80	RUNFREQ	O		RecordUpdateFrequency
Record type	VARCHAR	80	TYPE	O		RecordType
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Relational tables and views

Relational tables and views: Represents tables or views of relational databases.

The tag language for defining this object type is in the file FLGNYTAB.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.TABLES. For more information on table names, see “MDIS mappings” on page 159.

Table 93 provides information about the properties of the Relational tables and views object type.

Table 93. Properties of the Relational tables and views object type. The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		RecordLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short Description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long Description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
Catalog remarks	VARCHAR	254	REMARKS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database host server name	VARCHAR	80	SERVER	O		ServerName
Local database alias	CHAR	8	DBALIAS	O		ApplicationData

Table 93. Properties of the Relational tables and views object type (continued). The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Database or subsystem name	VARCHAR	80	DBNAME	R	1	DatabaseName
Table owner	VARCHAR	80	OWNER	R	2	OwnerName
Table name	VARCHAR	80	TABLES	R	3	RecordName
Base table owner name	CHAR	30	SRCOWNER	O		ApplicationData
Base table name	CHAR	128	SRCTBNAM	O		ApplicationData
Table data last refreshed	CHAR	26	FRESHDAT	O		RecordLastRefreshDate
Transformation program run mode	CHAR	30	RUNMODE	O		ApplicationData
Transformation program last run	CHAR	26	LASTRUN	O		ApplicationData
Transformation program run frequency	VARCHAR	80	RUNFREQ	O		RecordUpdateFrequency
Transformation program type	VARCHAR	32	SOURCE	O		ApplicationData
Partial or full table copy/update	CHAR	1	COPYCOMP	O		ApplicationData
Copied/updated data is in a consistent state	CHAR	1	CONSIST	O		ApplicationData
Catalog refresh/update frequency	VARCHAR	80	REFRESH	O		ApplicationData
Transformation program last changed	CHAR	26	PGMGEND	O		ApplicationData
Transformation program last compiled	CHAR	26	PGMCOMP	O		ApplicationData

Table 93. Properties of the Relational tables and views object type (continued). The MDIS name for this object type is Record.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Table type	VARCHAR	80	TYPE	O		RecordType
Definition represents a view	CHAR	1	TABLVIEW	O		ApplicationData
Internal name of table	CHAR	18	IDSINAME	O		ApplicationData
Table is used as a dimension table	CHAR	1	IDSDIM	O		ApplicationData
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by the Information Catalog Manager, R = required, O = optional						

Star Schemas: Represents relational data.

The tag language for defining this object type is in the file FLGNYSUB.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.STARSCHM. For more information on table names, see “MDIS mappings” on page 159.

Table 94 provides information about the properties of the Business subject areas object type.

Table 94. Properties of the Star Schemas object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Name	VARCHAR	80	NAME	R	1

Table 94. Properties of the Star Schemas object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
For further information . . .	VARCHAR	80	RESPNSBL	O	
URL to access data	VARCHAR	254	URL	O	
Timestamp source definition last changed	CHAR	26	SRCDATCF	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Subschemas: Represents logical groupings of records within a database.

The tag language for defining this object type is in the file FLGNYSUB.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.SUBSCHEM. For more information on table names, see "MDIS mappings" on page 159.

Table 95 provides information about the properties of the Subschemas object type.

Table 95. Properties of the Subschemas object type. The MDIS name for this object type is Subschema.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		SubschemaLongName

Table 95. Properties of the Subschemas object type (continued). The MDIS name for this object type is Subschema.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
For further information...	VARCHAR	80	RESPNSBL	O		ContactName
Database host server name	VARCHAR	80	SERVER	R	1	ServerName
Database or subsystem name	VARCHAR	80	DBNAME	R	2	DatabaseName
Subschema owner	VARCHAR	80	OWNER	O		OwnerName
Subschema name	VARCHAR	80	SSNAME	R	3	SubschemaName
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by the Information Catalog Manager, R = required, O = optional						

Transformations: Represents expressions or logic used to populate columns of data within the target relational database. Transformations objects indicate either the expression used to convert source operational data to target columns or the one-to-one mapping of source fields to target columns.

The tag language for defining this object type is in the file FLGNYFLT.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.FILTER. For more information on table names, see "MDIS mappings" on page 159.

Table 96 provides information about the properties of the Transformations object type.

Table 96. Properties of the Transformations object type. The MDIS name for this object type is Relationship.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Object type identifier	CHAR	6	OBJTYPID	S		
Instance identifier	CHAR	10	INSTIDNT	S		
Name	VARCHAR	80	NAME	R		RelationshipLongName
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		
Last Changed By	CHAR	8	UPDATEBY	S		
Short description	VARCHAR	250	SHRTDESC	O		BriefDescription
Long description	LONG VARCHAR	32700	LONGDESC	O		LongDescription
Actions	VARCHAR	254	ACTIONS	O		ApplicationData
Transformation program name	VARCHAR	80	FPNAME	R	1	ApplicationData
Transformation identifier	VARCHAR	254	FIDENT	R	2	RelationshipName
Transformation class or type	VARCHAR	80	TYPE	R	3	RelationshipType

Table 96. Properties of the Transformations object type (continued). The MDIS name for this object type is Relationship.

Property name	Data type	Size	Property short name	Value flag	UUI order	Maps to MDIS name:
Source column/field name, expression or parameters	LONG VARCHAR	32700	FEXPRESS	O		RelationshipExpression
Database host server name	VARCHAR	80	SERVER	O		ServerName
Transformation owner	VARCHAR	80	OWNER	O		OwnerName
Source sequence	CHAR	5	SRCSEQ	O		SourceSequenceOrder
Transformation ordinality	CHAR	5	ORDINAL	O		RelationshipOrdinality
Transformation bi-directionality	CHAR	1	DIRECT	O		RelationshipBidirectional
URL to access data	VARCHAR	254	URL	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
For further information...	VARCHAR	80	RESPNSBL			ContactName

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Elemental category

The Elemental category contains the following object types:

- "Audio clips" on page 193
- "Charts" on page 194
- "Documents" on page 195
- "Images or graphics" on page 196
- "Internet documents" on page 197
- "Lotus Approach queries" on page 198

- “Presentations” on page 199
- “Spreadsheets” on page 200
- “Text-based reports” on page 201
- “Video clips” on page 203

Audio clips: Represents files that contain audio information. These objects might represent electronic (AUD files) or printed (for example, CDs, tapes) audio information.

The tag language for defining this object type is in the file FLGNYAUD.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.AUDIO. For more information on table names, see “MDIS mappings” on page 159.

Table 97 provides information about the properties of the Audio clips object type.

Table 97. Properties of the Audio clips object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Audio clip filename	VARCHAR	254	FILENAME	R	1
Audio clip class or type	VARCHAR	80	TYPE	R	2

Table 97. Properties of the Audio clips object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Charts: Represents either printed or electronic charts.

The tag language for defining this object type is in the file FLGNYCHA.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.CHARTS. For more information on table names, see "MDIS mappings" on page 159.

Table 98 provides information about the properties of the Charts object type.

Table 98. Properties of the Charts object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Chart title	VARCHAR	254	TITLE	O	
Chart publication date	CHAR	26	RPRTDATE	O	

Table 98. Properties of the Charts object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Chart presentation format	VARCHAR	80	RPRTFRMT	O	
Chart presentation requirements	VARCHAR	254	DPPRESNT	O	
Chart owner	VARCHAR	80	OWNER	O	
Chart filename	VARCHAR	254	FILENAME	R	1
Chart class or type	VARCHAR	80	TYPE	R	2
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Documents: Represents books and technical papers. These publications might be printed or electronic, found locally or within a library.

The tag language for defining this object type is in the file FLGNYDOC.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.DOCS. For more information on table names, see "MDIS mappings" on page 159.

Table 99 provides information about the properties of the Documents object type.

Table 99. Properties of the Documents object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	

Table 99. Properties of the Documents object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Document author	VARCHAR	80	AUTHOR	R	1
Document location	VARCHAR	254	LOCATION	R	2
Document filename	VARCHAR	254	FILENAME	R	3
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Images or graphics: Represents graphic images, such as bitmaps.

The tag language for defining this or graphics object type is in the file FLGNYIMA.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.IMAGES. For more information on table names, see “MDIS mappings” on page 159.

Table 100 provides information about the properties of the Images or graphics object type.

Table 100. Properties of the Images or graphics object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	

Table 100. Properties of the Images or graphics object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Image filename	VARCHAR	254	FILENAME	R	1
Image class or type	VARCHAR	80	TYPE	R	2
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Internet documents: Represents Web sites and other documents on the Internet that might be of interest.

The tag language for defining this object type is in the file FLGNYINT.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.INTERNET. For more information on table names, see "MDIS mappings" on page 159.

Table 101 on page 198 provides information about the properties of the Internet documents object type.

Internet documents

Table 101. Properties of the Internet documents object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
URL to access data	VARCHAR	254	URL	R	1
Local filename	VARCHAR	254	FILENAME	R	2
Internet document class or type	VARCHAR	80	TYPE	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Lotus Approach queries: Represents Lotus Approach queries for available use with your organization's data.

The tag language for defining this object type is in the file FLGNYAPR.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.APPROACH. For more information on table names, see "MDIS mappings" on page 159.

Table 102 on page 199 provides information about the properties of the Lotus Approach queries object type.

Table 102. Properties of the Lotus Approach queries object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Approach object filename	VARCHAR	254	FILENAME	R	1
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Presentations: Represents printed or electronic presentations. These presentations might include product, customer, quality, and status presentations.

The tag language for defining this object type is in the file FLGNYPRE.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.PRESENT. For more information on table names, see “MDIS mappings” on page 159.

Table 103 on page 200 provides information about the properties of the Presentations object type.

Table 103. Properties of the Presentations object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Presentation filename	VARCHAR	254	FILENAME	R	1
Presentation class or type	VARCHAR	80	TYPE	O	
Presentation script	VARCHAR	254	SCRIPTFN	O	
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Spreadsheets: Represents desktop spreadsheets (for example, Lotus 1-2-3 or Microsoft Excel spreadsheets).

The tag language for defining this object type is in the file FLGNYSSH.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.SSHEETS. For more information on table names, see “MDIS mappings” on page 159.

Table 104 on page 201 provides information about the properties of the Spreadsheets object type.

Table 104. Properties of the Spreadsheets object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Spreadsheet class or type	VARCHAR	80	TYPE	O	
Spreadsheet filename	VARCHAR	254	FILENAME	R	1
Spreadsheet bitmap <captured> filename	VARCHAR	254	BITMAP	O	
URL to access data	VARCHAR	254	URL	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Text-based reports: Represents either printed or electronic reports.

The tag language for defining this object type is in the file FLGNYREP.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.REPORTS. For more information on table names, see “MDIS mappings” on page 159.

Table 105 on page 202 provides information about the properties of the Text–based reports object type.

Text-based reports

Table 105. Properties of the Text-based reports object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Report title	VARCHAR	254	TITLE	R	
Report publication date	CHAR	26	RPRTDATE	O	
Report presentation format	VARCHAR	80	RPRTFRMT	O	
Report presentation requirements	VARCHAR	254	DPPRESNT	O	
Report owner	VARCHAR	80	OWNER	O	
Report filename	VARCHAR	254	FILENAME	R	1
Report class or type	VARCHAR	80	TYPE	R	2
URL to access data	VARCHAR	254	URL	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Video clips: Represents files that contain video information. These objects might represent electronic (AVI files) or printed (for example, video tapes or laser disks) video information.

The tag language for defining this object type is in the file FLGNYVID.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.VIDEO. For more information on table names, see "MDIS mappings" on page 159.

Table 106 provides information about the properties of the Video clips object type.

Table 106. Properties of the Video clips object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Video clip filename	VARCHAR	254	FILENAME	R	1
Video clip class or type	VARCHAR	80	TYPE	R	2
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Contact category

The Contact category contains the “People to contact” object type.

People to contact: The People to contact object type identifies a person or group that is responsible for objects within the information catalog.

The tag language for defining the People to contact object type is in the file FLGNYCON.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.CONTACT. For more information on table names, see “MDIS mappings” on page 159.

Table 107 provides information about the properties of the People to contact object type.

Table 107. Properties of the People to contact object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	1
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Contact’s responsibility	VARCHAR	254	RESPONSE	R	2
Contact’s phone number	CHAR	15	PHONE	R	
Contact’s e-mail address	VARCHAR	254	EMAIL	R	

Table 107. Properties of the People to contact object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Contact's picture filename	VARCHAR	254	FILENAME	O	
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Dictionary category

The Dictionary category contains the "Glossary entries" object type.

Glossary entries: The Glossary entries object type represents definitions for terms used in the information catalog. Its properties are shown in Table 108.

The tag language for defining the Glossary entries object type is in the file FLGNYGLO.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.GLOSSARY. For more information on table names, see "MDIS mappings" on page 159.

Table 108 provides information about the properties of the Glossary entries object type.

Table 108. Properties of the Glossary entries object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	1
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	

Table 108. Properties of the Glossary entries object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Keywords	VARCHAR	254	KEYWORD	O	
Context of glossary definition	CHAR	32	CONTEXT	O	
Filename containing glossary definition	VARCHAR	254	FILENAME	O	
Glossary class or type	VARCHAR	80	TYPE	O	
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by the Information Catalog Manager, R = required, O = optional					

Support category

The Support category contains the following object types:

- "Information Catalog Manager news"
- "Online news services" on page 207
- "Online publications" on page 208

Information Catalog Manager news: The Information Catalog Manager news object type contains information about changes to the information catalog.

The tag language for defining the Information Catalog Manager news object type is in the file FLGNYDGN.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.DGNEWS. For more information on table names, see "MDIS mappings" on page 159.

Table 109 on page 207 provides information about the properties of the Information Catalog Manager news object type.

Table 109. Properties of the Information Catalog Manager news object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	1
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
News item date	CHAR	26	NEWSDATE	R	
News clip	VARCHAR	254	ABSTRACT	R	
Full news item	LONG VARCHAR	32700	NEWSITEM	O	
URL to access data	VARCHAR	254	URL	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Online news services: The Online news services object type represents news and information services that can be accessed online.

The tag language for defining the Online news services object type is in the file FLGNYOLN.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.OLNEWS. For more information on table names, see “MDIS mappings” on page 159.

Table 110 on page 208 provides information about the properties of the Online news services object type.

Table 110. Properties of the Online news services object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	1
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Service name	VARCHAR	254	SERVNAME	R	
URL to access data	VARCHAR	254	URL	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Online publications: The Online publications object type represents publications and other documents that can be accessed with online services.

The tag language for defining the Online publications object type is in the file FLGNYOLP.TYP in the \VWSWIN\DGWIN\TYPES directory.

The Information Catalog Manager DB2 storage table name for this object type is XXX.OLPUBS. For more information on table names, see “MDIS mappings” on page 159.

Table 111 on page 209 provides information about the properties of the Online publications object type.

Table 111. Properties of the Online publications object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	1
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Short description	VARCHAR	250	SHRTDESC	O	
Long description	LONG VARCHAR	32700	LONGDESC	O	
Actions	VARCHAR	254	ACTIONS	O	
Service name	VARCHAR	254	SERVNAME	R	
URL to access data	VARCHAR	254	URL	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Program category

The Program category can only contain the *Programs* object type.

The *Programs* object type is created when an information catalog is created. In the sample information catalog, DGV5SAMP, the *Programs* object type is named *Programs that can be invoked from Information Catalog Manager objects*.

Programs that can be invoked from Information Catalog Manager objects: Used to define an application that is capable of processing a particular object type.

The Information Catalog Manager DB2 storage table name for this object type is XXX.GLOSSARY. For more information on table names, see “MDIS mappings” on page 159.

Table 112 on page 210 provides information about the properties of the *Programs that can be invoked from Information Catalog Manager objects* object type.

Programs that can be invoked from Information Catalog Manager objects

Table 112. Properties of the "Programs that can be invoked from Information Catalog Manager objects" object type

Property name [1]	Data type	Size	Property short name	Value flag [2]	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Class	CHAR	25	UUICLASS	R	1
Qualifier 1	VARCHAR	48	UUIQUAL1	R	2
Qualifier 2	VARCHAR	48	UUIQUAL2	R	3
Qualifier 3	VARCHAR	48	UUIQUAL3	R	4
Identifier	VARCHAR	70	UUIDENT	R	5
Object type this program handles	CHAR	8	HANDLES	O	
Start by invoking	VARCHAR	250	STARTCMD	R	
Parameter list is	VARCHAR	1800	PARMLIST	O	
Short description	VARCHAR	250	SHRTDESC	O	

Note:

1. Descriptions and examples of the required properties are in *Information Catalog Manager Administration Guide*.
2. S = generated by the Information Catalog Manager, R = required, O = optional

Attachment category

The Attachment category can contain only the "Comments" object type.

The Comments object type is created when an information catalog is created.

Comments: Used to comment on other objects in the information catalog.

Table 113 provides information about the properties of the Comments object type.

Table 113. Properties of the Comments object type

Property name	Data type	Size	Property short name	Value flag	UUI order
Object type identifier	CHAR	6	OBJTYPID	S	
Instance identifier	CHAR	10	INSTIDNT	S	
Name	VARCHAR	80	NAME	R	1
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S	
Last Changed By	CHAR	8	UPDATEBY	S	
Creator	CHAR	8	CREATOR	R	2
Creation time stamp	TIMESTAMP	26	CREATSTP	R	3
Status	CHAR	80	STATUS	O	
Actions	VARCHAR	250	ACTIONS	O	
Extra Information	VARCHAR	80	EXTRA	O	
Long Description	LONG VARCHAR	32700	LONGDESC	O	

Note: S = generated by the Information Catalog Manager, R = required, O = optional

Predefined program objects

Program object types shown in Table 114 are provided in the sample information catalog. The table also shows the property name that you use to associate with the Information Catalog Manager program object when launching a program.

Table 114. Generic predefined program objects in the sample information catalog

Type of information	Program name	Object type	Property name
Multimedia files	Microsoft Media Player	Audio clips	Audio clip filename
	Microsoft Media Player	Business subject areas	Filename
	Microsoft Media Player	Presentations	Presentation filename
	Microsoft Media Player	Video clips	Video clip filename

Table 114. Generic predefined program objects in the sample information catalog (continued)

Type of information	Program name	Object type	Property name
Bitmap files	Microsoft Paint	Images or graphics	Graphic filename
	Microsoft Paint	People to contact	Contact's picture filename
Spreadsheet files	Microsoft Excel	Spreadsheets	Spreadsheet filename
	Microsoft Paint	Spreadsheets	Spreadsheet filename
	Lotus 1-2-3	Spreadsheets	Spreadsheet filename
Web pages	Netscape Navigator	Online news	URL to access data
	Netscape Navigator	Online publications	URL to access data
	Microsoft Internet Explorer	Internet documents	URL to access data
	Microsoft Internet Explorer	Online news	URL to access data
	Microsoft Internet Explorer	Online publications	URL to access data

Table 115 lists specific IBM Business Partners who have applications that are integrated with the Information Catalog Manager. The information in this table similar to that in Table 114 on page 211.

Table 115. Predefined program objects in sample information catalog — IBM Business Partners

Type of information	Program name	Object type	Property name
Lotus	Approach	Lotus Approach	Approach object filename
	Freelance Graphics	Presentations	Presentation object filename
Hyperion	Lotus 1-2-3 with Essbase Spreadsheet add-in	Spreadsheets	Spreadsheet filename
	Microsoft Excel with Essbase Spreadsheet add-in	Spreadsheets	Spreadsheet filename
Brio	Brio Query	Text based reports	Report filename
	Netscape Navigator (use with Brio.Insights plug-in)	Text based reports	URL to access data
	Microsoft Internet Explorer (use with Brio.Insights plug-in)	Text based reports	URL to access data

Table 115. Predefined program objects in sample information catalog — IBM Business Partners (continued)

Type of information	Program name	Object type	Property name
BusinessObjects	BusinessObjects	Databases	None
	BusinessObjects	Text based reports	Report filename
	Microsoft Excel (used with BusinessQuery add-in)	Spreadsheets	Spreadsheet filename
	Microsoft Internet Explorer (used to access WebIntelligence Java® applet)	Internet documents	URL to access data
	Netscape Navigator (used to access WebIntelligence Java applet)	Internet documents	URL to access data
Cognos	PowerPlay	Text-based reports	Report filename
	Impromptu	Text-based reports	Report filename
	Microsoft Internet Explorer (used with Impromtu Web Query)	Internet documents	URL to access data
	Netscape Navigator (used with Impromptu Web Query)	Internet documents	URL to access data
	Netscape Navigator (used to access PowerPlay Web edition HTML pages)	Internet documents	URL to access data
Wired for OLAP	Wired for OLAP View	Text-based reports	configure Default user login, and Startup options
	Wired for OLAP Home Page within Netscape	Text-based reports	configure Default user login, and Startup options
	Wired for OLAP Home Page within Microsoft Internet Explorer	Text-based reports	configure Default user login, and Startup options
Seagate	Crystal Reports	Text-based reports	Report filename
Microsoft Access	Microsoft Access	Database	
Microsoft PowerPoint	Microsoft PowerPoint Viewer	Text-based reports	Report filename

Predefined program objects

Table 115. Predefined program objects in sample information catalog — IBM Business Partners (continued)

Type of information	Program name	Object type	Property name
	Microsoft PowerPoint Viewer within Netscape	Text-based reports	URL to access data
	Microsoft PowerPoint Viewer within Microsoft Internet Explorer	Text-based reports	URL to access data

Chapter 9. Tag language

The Information Catalog Manager tag language allows you to format your descriptive data so that you can import it into your information catalog. The tag language tells the Information Catalog Manager what to do with the descriptive data that it imports. The Information Catalog Manager also exports descriptive data into tag language files so that you can back up your information catalog or transfer data from one information catalog to another.

By formatting descriptive data with the tag language, you can move descriptive data from one information catalog to another and define Information Catalog Manager object types and objects. You can also write and use extract programs to extract descriptive data from other sources, such as a relational database catalog, that you can import to your information catalog Table 116 shows the tags in the tag language and the actions that these tags perform.

Table 116. Information Catalog Manager tags

Task	Tag names	For details
Record diskette sequence	DISKCNTL	see page 230
Identify action to be taken on input data	ACTION.OBJINST ACTION.OBJTYPE ACTION.RELATION	see page 218 see page 223 see page 227
Describe data to the information catalog	OBJECT PROPERTY INSTANCE RELTYPE	see page 237 see page 242 see page 231 see page 246
Identify when changes are committed and where check point occurs	COMMIT	see page 229
Identify user comments	COMMENT	see page 228
Format data	NL TAB	see page 236 see page 247

Rules for writing tag language files

The rules explained in this section apply to all tag language files.

- Each tag name must start with a colon and end with a period. Do not put spaces between the colon and the tag name, or between the tag name and the period. For example:

```
:ACTION.OBJINST.
```

The tag name must be one of the tag names that are listed in Table 116 on page 215.

- Include at least one keyword with all tags except COMMENT, NL, or TAB.
- Write the keyword and its value like this:

  ```
  keyword(value)
  ```
- Specify keywords in any order. The only exception is that the SOURCEKEY keyword of the INSTANCE tag must be the first keyword.
- Use a blank to separate keywords.
- Enclose in parentheses the value of a keyword. If the value contains a parenthesis, enclose the parenthesis in a pair of apostrophes; for example:

  ```
  keyword(value'('1')')
  ```
- Do not use OBJTYPID, INSTIDNT, UPDATIME, or UPDATEBY as property short names (*short_names*) with the PROPERTY or INSTANCE tags.
- These property names are reserved by Information Catalog Manager:

  OBJTYPID
  INSTIDNT
  NAME
  UPDATIME
  UPDATEBY

  You can specify NAME as the *short_name* on the PROPERTY tag if you identify NAME as a UUI property for an object type when using ACTION.OBJTYPE(ADD) or ACTION.OBJTYPE(MERGE), as shown:

  ```
  :PROPERTY.SHRTNAME(NAME) UUISEQ(1)
  ```

How the Information Catalog Manager reads tag language files

When you code a tag language file, consider how the Information Catalog Manager reads tag language files:

- The Information Catalog Manager reads the entire tag language file as a continuous data stream.
- The Information Catalog Manager treats any character with a hexadecimal value under X'20' (except for tab and new line character tags that are specified in property values) as a control character and ignores that character.
- The Information Catalog Manager considers a tag complete when it encounters the next tag in the tag language file.
- Tags and keywords are not translated into national languages.
- Only the values for the keywords in Table 117 on page 217 are enabled for double-byte character set (DBCS) support.

Table 117. Keyword values enabled for DBCS

Tag name	Keywords	Variable value
OBJECT	EXTNAME ICWFILE	*ext_name* *Windows_ICON_file_name*
PROPERTY	EXTNAME	*ext_name*
COMMIT	CHKPID	*checkpt_id*
INSTANCE	*UUI_short_name* *or* *short_name*	*UUI_property_value* *or* *property_value*

All user-defined property values can use DBCS characters.

- The Information Catalog Manager accepts DBCS blanks only in the keyword values that are shown in Table 118. If DBCS blanks appear anywhere else in the tag language file, errors can occur.

Table 118. Keyword values enabled for DBCS blank characters

Tag name	Keywords
ACTION	OBJTYPE OBJINST RELATION
OBJECT	All keywords
PROPERTY	All keywords
RELTYPE	All keywords
COMMIT	CHKPID
INSTANCE	*UUI_short_name* *or* *short_name*

Valid data types for Information Catalog Manager descriptive data

Table 119 shows the valid data types for Information Catalog Manager descriptive data.

Table 119. Valid data types for Information Catalog Manager descriptive data

Data type	Description
CHAR	Fixed-length character string between 1 and 254 bytes long. Pad the value on the right with trailing blanks if the value is shorter than the defined data length for the property.
TIMESTAMP	26-character timestamp in the following format: `yyyy-mm-dd-hh.mm.ss.nnnnnn`

Table 119. Valid data types for Information Catalog Manager descriptive data (continued)

Data type	Description
LONG VARCHAR	Long varying-length character string between 1 and 32 700 bytes long. You cannot specify a property with a data type of LONG VARCHAR as a UUI property.
VARCHAR	Varying-length character string between 1 and 4 000 bytes long.

The Information Catalog Manager automatically removes trailing blanks from variable values and adjusts their length accordingly before validating and accepting the request.

If a required value is not specified or contains all blanks, the Information Catalog Manager inserts the values that are shown in Table 120.

Table 120. Information Catalog Manager-supplied values

Data type	Supplied value
CHAR	A not-applicable symbol as the first character and padded with trailing blanks to fill the defined length.
TIMESTAMP	9999-12-31-24.00.00.000000
LONG VARCHAR	A not-applicable symbol
VARCHAR	A not-applicable symbol

How to read the tag language syntax diagrams

Code the tags and keywords exactly as they are shown in the text. The tags and keywords are represented like this:

```
:tagname.keyword() keyword()
```

Valid values that you can substitute for variables are described in the keyword list. The values are represented like this: *variable*

In tag descriptions, a vertical bar in each pair of keywords or values indicates that you must include one of the pair with the tag. For example, the syntax for the PROPERTY tag includes the NULLS keyword values NULLS(Y|N). You must code either NULLS(Y) or NULLS(N).

ACTION.OBJINST

Identifies the action to be performed on the object that is described with the tags that follow the ACTION tag.

Context

ACTION.OBJINST is used to create, delete, or maintain Information Catalog Manager objects.

ACTION.OBJINST is followed by one or more OBJECT and INSTANCE tags, which define the object to act on.

Syntax

```
:ACTION.OBJINST(option)
```

Options

The following options are valid for ACTION.OBJINST:

ADD
DELETE
DELETE_TREE_ALL
DELETE_TREE_REL
MERGE
UPDATE

ACTION.OBJINST(ADD)

Adds an object.

Context:

```
:ACTION.OBJINST(ADD)
:OBJECT.TYPE()
:INSTANCE.short_name()
:INSTANCE.short_name()

:OBJECT.TYPE()
:INSTANCE.short_name()
:INSTANCE.short_name()
```

Figure 33. Using the ACTION.OBJINST tag when adding objects

Rules:

- The object must not already exist.
- Both the OBJECT tag and the INSTANCE tag must follow the ACTION.OBJINST(ADD) tag.
  – The OBJECT tag identifies the object type for the new object.
  – The INSTANCE tag specifies the property values for the new object.
- One or more INSTANCE tags can follow a single OBJECT tag, if the objects are for the same object type.

- One or more sets of an OBJECT tag with INSTANCE tags can follow an ACTION.OBJINST(ADD) tag to describe objects of different object types to add.

ACTION.OBJINST(DELETE)

Deletes an object.

Context:

```
:ACTION.OBJINST(DELETE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)

:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 34. Using the ACTION.OBJINST tag when deleting objects

Rules:

- The specified object must already exist.
- Both the OBJECT tag and the INSTANCE tag must follow the ACTION.OBJINST(DELETE) tag.
  - The OBJECT tag identifies the object type for the object to be deleted.
  - The INSTANCE tag specifies the UUI property values for the object to be deleted.
- One or more INSTANCE tags can follow a single OBJECT tag, if the objects are for the same object type.
- One or more sets of an OBJECT tag with INSTANCE tags can follow an ACTION.OBJINST(DELETE) tag to describe objects of different object types to delete.
- If the object to delete is a Grouping object, it cannot contain another object. If it does, the delete fails. Use ACTION.OBJINST(DELETE_TREE_ALL) instead.

ACTION.OBJINST(DELETE_TREE_ALL)

Deletes a Grouping category object, all Comments objects that are attached to it, and all ATTACHMENT, CONTACT, and LINK relationships in which it participates. Deletes all objects that are contained in the Grouping category object, all Comments objects attached to them, and all ATTACHMENT, CONTACT, and LINK relationships in which they participate.

Context:

```
:ACTION.OBJINST(DELETE_TREE_ALL)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)

:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 35. Using the ACTION.OBJINST tag when deleting Grouping category objects and contained objects

Rules:

- The specified object must already exist and be a Grouping category object.
- Both the OBJECT tag and the INSTANCE tag must follow the ACTION.OBJINST(DELETE_TREE_ALL) tag.
  - The OBJECT tag identifies the object type for the object to delete.
  - The INSTANCE tag specifies the UUI property values for the object that is being deleted.
- One or more INSTANCE tags can follow a single OBJECT tag, if the objects are for the same object type.
- One or more sets of an OBJECT tag with INSTANCE tags can follow an ACTION.OBJINST(DELETE_TREE_ALL) tag to describe objects of different object types to be deleted.

ACTION.OBJINST(DELETE_TREE_REL)

Deletes a Grouping category object, all Comments objects attached to it, and all ATTACHMENT, CONTACT, CONTAIN, and LINK relationships in which it participates.

Context:

```
:ACTION.OBJINST(DELETE_TREE_REL)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)

:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 36. Using the ACTION.OBJINST tag when deleting Grouping category objects and relationships

Rules:

- The specified object must already exist and be a Grouping category object.
- Both the OBJECT tag and the INSTANCE tag must follow the ACTION.OBJINST(DELETE_TREE_REL) tag.

- – The OBJECT tag identifies the object type for the object being deleted.
- – The INSTANCE tag specifies the UUI property values for the object being deleted.
- One or more INSTANCE tags can follow a single OBJECT tag, if the objects are for the same object type.
- One or more sets of an OBJECT tag with INSTANCE tags can follow an ACTION.OBJINST(DELETE_TREE_REL) tag to describe objects of different object types to be deleted.

ACTION.OBJINST(MERGE)

Searches for the input object's UUI in the information catalog to see whether the input object exists.

If the object exists, the Information Catalog Manager updates the property values of the object in the information catalog. If the object does not exist, the Information Catalog Manager creates a new object.

Context:

```
:ACTION.OBJTYPE(MERGE)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()

:ACTION.OBJINST(MERGE)
:OBJECT.TYPE()
:INSTANCE.short_name()
```

Figure 37. Using the ACTION.OBJINST tag when merging objects

Rules:

- If the object exists, the Information Catalog Manager updates the property values of the object in the information catalog. If the object does not exist, the Information Catalog Manager creates a new object.
- Both the OBJECT tag and the INSTANCE tag must follow the ACTION.OBJINST(MERGE) tag.
  - – The OBJECT tag identifies the object type for the object being merged.
  - – The INSTANCE tag specifies the property values for the object being merged.
- You must have an ACTION.OBJTYPE(MERGE) tag for a given object type earlier in the tag language file than the ACTION.OBJINST(MERGE) tag for the same object type. This ensures that the object type exists in the information catalog that you are importing to before the Information Catalog Manager can add or update (merge) objects.

  You cannot use ACTION.OBJTYPE(MERGE) for an object type that belongs to the Program or Attachment categories, because you cannot create new

Program or Attachment object types. However, you can use ACTION.OBJINST(MERGE) with Program objects, without specifying the ACTION.OBJTYPE(MERGE) first.

ACTION.OBJINST(UPDATE)
Updates the value of an object.

Context:

```
:ACTION.OBJINST(UPDATE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) short_name()
```

Figure 38. Using the ACTION.OBJINST tag when updating objects

Rules:

- The specified object must already exist.
- Both the OBJECT tag and the INSTANCE tag must follow the ACTION.OBJINST(UPDATE) tag.
  - The OBJECT tag identifies the object type for the object being updated.
  - The INSTANCE tag specifies the UUI property values, which identify the object to be updated, and the property values that are being updated.

  Only the property values specified on the INSTANCE tag are updated.

ACTION.OBJTYPE

Identifies the action to perform on the object type that is described with the tags that follow ACTION.OBJTYPE.

Context

ACTION.OBJTYPE is used to create, delete, or maintain Information Catalog Manager object types.

ACTION.OBJTYPE is followed by one or more OBJECT and PROPERTY tags, which define the object type being acted on.

Syntax

```
:ACTION.OBJTYPE(option)
```

Options

The following options are valid with ACTION.OBJTYPE:
ADD
APPEND
DELETE

DELETE_EXT
MERGE
UPDATE

ACTION.OBJTYPE(ADD)

Creates the object type.

Context:

```
:ACTION.OBJTYPE(ADD)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 39. Using the ACTION.OBJTYPE tag when adding object types

Rules:

- The object type must not exist.
- An OBJECT tag and its associated PROPERTY tags must immediately follow the ACTION.OBJTYPE(ADD) tag.
  - The OBJECT tag defines the attributes of the new object type.
  - The PROPERTY tags define the properties that belong to the new object type. The Information Catalog Manager automatically defines the following required properties for every object type:
    OBJTYPID
    INSTIDNT
    NAME
    UPDATIME
    UPDATEBY
- You cannot add object types that belong to the Program or Attachment categories.

ACTION.OBJTYPE(APPEND)

Appends a property to an existing object type.

Context:

```
:ACTION.OBJTYPE(APPEND)
:OBJECT.TYPE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 40. Using the ACTION.OBJTYPE tag when adding properties to object types

Rules:

- The object type must exist.

- The property being appended must not exist.
- Do not assign the property a UUISEQ value other than 0 (the default). Appended properties must be optional with NULLS(Y) and cannot be part of the UUI.
- An OBJECT tag and one or more PROPERTY tags must immediately follow the ACTION.OBJTYPE(APPEND) tag.
  - The OBJECT tag identifies the object type being appended.
  - Each PROPERTY tag defines a property being appended.
- You cannot append to object types that belong to the Attachment category.

ACTION.OBJTYPE(DELETE)

Deletes the object type.

Context:

```
:ACTION.OBJTYPE(DELETE)
:OBJECT.TYPE()
```

Figure 41. Using the ACTION.OBJTYPE tag when deleting object types

Rules:

- The object type must exist. No objects of the object type can exist.
- One or more OBJECT tags must follow an ACTION.OBJTYPE(DELETE) tag. Each OBJECT tag identifies the object type being deleted.
- You cannot delete object types that belong to the Program or Attachment categories.

ACTION.OBJTYPE(DELETE_EXT)

Deletes the object type and objects of that object type.

Context:

```
:ACTION.OBJTYPE(DELETE_EXT)
:OBJECT.TYPE()
```

Figure 42. Using the ACTION.OBJTYPE tag when deleting object types and all objects of that type

Rules:

- The object type must exist.
- The object cannot contain objects of a different object type.
- One or more OBJECT tags must follow the ACTION.OBJTYPE(DELETE) tag. Each OBJECT tag identifies the object type being deleted.

- You cannot delete object types that belong to the Program or Attachment categories.

ACTION.OBJTYPE(MERGE)

Checks the information catalog for the input object type name to see if the object type exists.

If the object type exists, the Information Catalog Manager compares properties of the input object type to the properties of the stored object type. If the properties match, then the object types are treated as identical; if not, the input object type is not valid.

If the object type does not exist, the Information Catalog Manager creates a new object type.

Context:

```
:ACTION.OBJTYPE(MERGE)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()

:ACTION.OBJINST(MERGE)
:OBJECT.TYPE()
:INSTANCE.short_name()
```

Figure 43. Using the ACTION.OBJTYPE tag when merging object types

Rules:

- An OBJECT tag and its associated PROPERTY tags must immediately follow the ACTION.OBJTYPE(MERGE) tag.
  - The OBJECT tag defines the object type being merged.
  - Each PROPERTY tag defines a property that belongs to the object type.
- Before you can merge objects, you must merge object types to ensure that a valid object type exists in the target information catalog. Therefore, an ACTION.OBJTYPE(MERGE) tag must appear before an ACTION.OBJINST(MERGE) tag in the tag language file.
- You cannot merge object types that belong to the Program or Attachment categories.

ACTION.OBJTYPE(UPDATE)

Changes an object-type external name and ICON file information.

Context:

```
:ACTION.OBJTYPE(UPDATE)
:OBJECT.TYPE() EXTNAME() ICOFILE() ICWFILE()
```

Figure 44. Using the ACTION.OBJTYPE tag when updating object types

Rules:
- The object type must already exist.
- One or more OBJECT tags must follow the ACTION tag.

ACTION.RELATION

Identifies the action to perform on the relationship that is described with the tags that follow ACTION.RELATION.

Context

ACTION.RELATION is used to create or delete information catalog relationships.

ACTION.RELATION is followed by one or more RELTYPE and INSTANCE tags, which define the relationships being acted on.

Syntax

```
:ACTION.RELATION(option)
```

Options

The following options are valid with ACTION.RELATION:
  ADD
  DELETE

ACTION.RELATION(ADD)

Defines an ATTACHMENT, CONTACT, CONTAIN, or LINK relationship.

Context:

```
:ACTION.RELATION(ADD)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 45. Using the ACTION.RELATION tag when adding relationships

Rules:
- If the specified relationship does not exist, the relationship is added. If the specified relationship exists, the Information Catalog Manager writes an informational message and continues processing.

- A RELTYPE tag and one or more INSTANCE tags must immediately follow the ACTION.RELATION(ADD) tag.
  - The RELTYPE tag defines the type of relationship that is being added and specifies the object types of the objects to associate.
  - Each INSTANCE tag specifies the UUI property values that identify the two objects that are being associated.

ACTION.RELATION(DELETE)

Deletes a relationship.

Context:

```
:ACTION.RELATION(DELETE)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 46. Using the ACTION.RELATION tag when deleting relationships

Rules:

- The relationship is deleted if it exists; otherwise, the Information Catalog Manager writes an informational message and continues processing.
- A RELTYPE tag and one or more INSTANCE tags must immediately follow the ACTION.RELATION(DELETE) tag.
  - The RELTYPE tag defines the type of relationship that is being deleted and specifies the object types of the associated objects.
  - Each INSTANCE tag specifies the UUI property values that identify the two associated objects.

COMMENT

Identifies comments in the tag language file. Place this tag between any complete tag specifications in your file.

The Information Catalog Manager ignores comments when importing a tag language file.

Syntax

```
:COMMENT.your comments
```

```
:COMMENT.This is the text of a comment.
```

Figure 47. Example of a COMMENT tag

Rules

- You cannot place a COMMENT tag between another tag and its keywords or between keywords.
- The comment text must not contain any Information Catalog Manager tags (for example `:ACTION.`), because each tag ends either at the end of the file or at the beginning of the next valid tag.

COMMIT

Identifies a commit point. Requests that the Information Catalog Manager commit the current changes to the database.

If the Information Catalog Manager encounters an error while importing a tag language file, it rolls back all changes that are made to the information catalog since the last time changes were committed.

Include COMMIT checkpoints at regular intervals so that you import Information Catalog Manager tag language files more efficiently.

Including COMMIT checkpoints before and after defining or deleting object types, sets of objects, and sets of relationships can help maintain the integrity of your descriptive data.

Regular COMMIT checkpoints limit the number of changes that the Information Catalog Manager cancels when it rolls back the information catalog.

Frequent COMMIT checkpoints make the echo file easier to read if there are errors in the tag language file. When the COMMIT tag is processed successfully, the Information Catalog Manager clears the echo file of the tags that were processed before the COMMIT tag. The echo file then contains only tags that describe uncommitted changes.

Context

Place this tag after one or more complete action specifications (a set of ACTION, OBJECT, RELTYPE, and INSTANCE tags).

Syntax

```
:COMMIT.CHKPID(checkpt_id)
```

```
:COMMIT.CHKPID(Added_relationships)
```

Figure 48. Example of a COMMIT tag

Keywords

CHKPID
Required keyword.

checkpt_id
An identifier that the Information Catalog Manager saves when it processes a COMMIT tag.

If the import of a tag language file fails after a COMMIT tag processes successfully, you need to import the rest of the tag language file starting at the last checkpoint. This option is available with the import function. The Information Catalog Manager uses the stored *checkpt_id* to locate the proper COMMIT tag.

The value of *checkpt_id* must be unique within each tag language file. Otherwise, the results of restart processing are unpredictable.

The maximum length of *checkpt_id* is 26 characters.

checkpt_id is not case-sensitive.

Rules

Specify a COMMIT tag when the data is consistent.

To prevent the target information catalog transaction log from filling up, specify COMMIT tags at regular intervals in the tag language file.

An ACTION tag must follow the COMMIT tag, if additional data in the same tag language file needs to be processed.

DISKCNTL

Identifies the diskette sequence number when the tag language file is stored on one or more diskettes.

Context

When one tag language file is stored on one or more diskettes, DISKCNTL is the first tag on each diskette.

Syntax

:DISKCNTL.SEQUENCE(*nn***, + | -)**

```
:DISKCNTL.SEQUENCE(01,+)
```

Figure 49. Example of a DISKCNTL tag for the first of a sequence of diskettes

Keywords

SEQUENCE
Required keyword

nn	A one-digit or two-digit number that indicates the number of the diskette in sequence. The first number for any sequence of disks must be `1` or `01`. This value increases by 1 for subsequent diskettes. The numbers for a set of three diskettes are `1`, `2`, `3`, or `01`, `02`, `03`.
+	Additional diskettes containing the tag language file follow this one.
–	The last or only diskette that contains the tag language file.

Rules

If this tag is specified, it must be the first tag in each tag language file. If the tag is missing and the tag language file is on diskette, the import program assumes that the tag language file is contained on one diskette.

If a tag language file is stored on the hard disk, this tag is not applicable. If the tag is present, it is ignored.

INSTANCE

Defines or identifies objects or relationships to be acted on.

Context

This tag is required following:

:ACTION.OBJINST	The INSTANCE tag follows an OBJECT tag.
:ACTION.RELATION	The INSTANCE tag follows a RELTYPE tag.

Syntax

There are four formats for the INSTANCE tag, depending on the format of the ACTION tag:

ACTION.OBJINST(ADD) or ACTION.OBJINST(MERGE)

Adding or merging objects

```
:INSTANCE.short_name (property_value) . . .
```

Context:

```
:ACTION.OBJINST(ADD)
:OBJECT.TYPE()
:INSTANCE.short_name()
```

Figure 50. Using the INSTANCE tag when adding objects

```
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE()
:INSTANCE.short_name()
:short_name()
:short_name()
```

Figure 51. Using the INSTANCE tag when merging objects

Keywords:

short_name
: Identifies each property by its 8-character short name. This value is not case sensitive; you can specify this value by using uppercase or lowercase characters. If an INSTANCE tag has multiple short names associated with it, use only one INSTANCE tag followed the short names as shown in Figure 51.

property_value
: Specifies the value of the property for the given object. This value is case sensitive.

Rules:

- When adding an object:
  - You must specify all UUI values, a value for the NAME property, and values for any other properties that are defined as required.
  - You can omit a property that does not have a value to add from the INSTANCE tag. However, if an omitted property is a required property with a CHAR, VARCHAR, or LONG VARCHAR data type, a not-applicable symbol is generated and stored in the information catalog. If an omitted required property has a TIMESTAMP data type, then the Information Catalog Manager generates and stores the value 9999-12-31-24.00.00.000000.
- When merging an object:
  - You must specify all UUI values, to ensure that matching objects can be identified.

– You can omit a property that does not have a value to be added or updated. However, if the defined object does not exist, and the omitted property is required, then a not-applicable symbol is generated and stored in the information catalog.

ACTION.OBJINST(DELETE) or ACTION.OBJINST(DELETE_TREE_ALL) or ACTION.OBJINST(DELETE_TREE_REL)

Deleting an object

:INSTANCE.SOURCEKEY(*UUI_short_name* **(***UUI_property_value***) . . .)**

Context:

```
:ACTION.OBJINST(DELETE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 52. Using the INSTANCE tag when deleting objects

```
:ACTION.OBJINST(DELETE_TREE_ALL)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 53. Using the INSTANCE tag when deleting Grouping category objects and contained objects

```
:ACTION.OBJINST(DELETE_TREE_REL)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 54. Using the INSTANCE tag when deleting Grouping category objects and relationships

Keywords:

SOURCEKEY
Specifies the UUI property values that identify a particular object.

SOURCEKEY must be the first keyword of the INSTANCE tag.

UUI_short_name
Identifies a UUI property name by its 8-character short name. Specify all of the *UUI_short_name*(*UUI_property_value*) combinations. The *UUI_short_name* is not case sensitive; you can specify this value by using uppercase or lowercase characters.

UUI_property_value
Specifies the value of a UUI property for a particular object. This value is case sensitive.

Rules: You must specify one *UUI_short_name*(*value*) combination for each property that is defined as a UUI property for the object type. Each object type has one or more properties defined as UUI properties. These properties uniquely identify an object in the information catalog.

ACTION.OBJINST(UPDATE)

Updating property values for an object

```
:INSTANCE.SOURCEKEY(UUI_short_name (UUI_property_value) . . . )
                    short_name (property_value) . . .
```

Context:

```
:ACTION.OBJINST(UPDATE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) short_name()
```

Figure 55. Using the INSTANCE tag when updating objects

Keywords:

SOURCEKEY
Specifies the UUI property values that identify a particular object.

SOURCEKEY must be the first keyword of the INSTANCE tag.

UUI_short_name
Identifies a UUI property by its 8-character short name. The *UUI_short_name* is not case sensitive; you can specify this value by using uppercase or lowercase characters.

UUI_property_value
This value is case sensitive. With *UUI_short_name*, specifies the value of a UUI property for a particular object.

short_name
Identifies the property to be updated by its 8-character short name. The *short_name* is not case sensitive; you can specify this value by using uppercase or lowercase characters.

You cannot specify the following property short names because you cannot update these properties: OBJTYPID, INSTIDNT, UPDATIME, UPDATEBY.

property_value
With *short_name*, specifies the new value of the property for the given object. This value is case sensitive.

Rules: You must specify one *UUI_short_name*(*value*) combination for each property that is defined as a UUI property for the object type. Each object type has one or more properties defined as UUI properties. These properties uniquely identify an object in the information catalog.

If you specify a property value, that value is updated in the information catalog. If you do not specify a property value, the value is not updated.

ACTION.RELATION(ADD) or ACTION.RELATION(DELETE)

Adding or deleting relationships

```
:INSTANCE.SOURCEKEY(UUI_short_name (UUI_property_value)...)
               TARGETKEY(UUI_short_name (UUI_property_value)...)
```

Context:

```
:ACTION.RELATION(ADD)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 56. Using the INSTANCE tag when adding relationships

```
:ACTION.RELATION(DELETE)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 57. Using the INSTANCE tag when deleting relationships

Keywords:

SOURCEKEY
Specifies the UUI property values that identify the first object in a relationship.

When the relationship is:	The SOURCEKEY identifies:
Contains	The Grouping category object
Contact	The object the contact is for
Attachment	The object the comment is for
Link	Either object to link

SOURCEKEY must be the first keyword of the INSTANCE tag.

TARGETKEY
Specifies the UUI property values that identify the second object in a relationship.

When the relationship is:	The TARGETKEY identifies:
Contains	The Elemental category object

Contact	The Contact category object
Attachment	The Attachment category object
Link	Either object to link

TARGETKEY must be the second keyword of the INSTANCE tag.

UUI_short_name
Identifies a UUI property name by its 8-character short name. This value is not case sensitive; you can specify this value by using uppercase or lowercase characters.

UUI_property_value
Specifies the value of a UUI property for a particular object. This value is case sensitive.

Rules: For each object, you must specify one *UUI_short_name*(*value*) combination for each property that is defined as a UUI property for the object type. Each object type has one or more properties defined as UUI properties. These properties uniquely identify an object in the information catalog.

You must separate each *UUI_short_name*(*value*) and *short_name*(*value*) pair with a blank, as shown in Figure 58.

```
:INSTANCE.SOURCEKEY(UUIname1(value1) UUIname2(value2)) sname3(value3) sname4(value4)
```

Figure 58. Example of an INSTANCE tag with several short names

Leading blanks that are included between the parentheses for a value become part of the value; trailing blanks are removed. The Information Catalog Manager counts these blanks as part of the data length when determining whether the length of the value is valid. An error occurs if you include extra leading blanks or trailing blanks on a value that make the entire value longer than the maximum allowed length.

NL

Specifies a new line within a property value.

The Information Catalog Manager manager reads only NL tags that are specified within non-UUI property values and ignores all others.

Syntax

```
:NL.
```

Rules

Use NL tags only within the specification of *property_values* in INSTANCE tags.

OBJECT

Defines the attributes for an object type or identifies an object type.

Context

This tag is required immediately following:
ACTION.OBJTYPE
ACTION.OBJINST

Syntax

```
:OBJECT.TYPE(type) CATEGORY(category) EXTNAME(ext_name)
        PHYNAME(table_name) ICOFILE( )
           ICWFILE(Windows_ICON_file_name)
```

Different OBJECT tag keywords are required or valid depending on the type of ACTION tag the OBJECT tag follows.

ACTION.OBJTYPE(ADD) or ACTION.OBJTYPE(MERGE)

Adding or merging object types

Context:

```
:ACTION.OBJTYPE(ADD)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 59. Using the OBJECT tag when adding object types

```
:ACTION.OBJTYPE(MERGE)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 60. Using the OBJECT tag when merging object types

Keywords:

TYPE
Specifies the name of an object type.

Required keyword.

type
Defines and identifies the short name for a specific object type.

The value of *type* must be unique to an object type across all related information catalogs that contain the same object type. This ensures that objects of this object type can be shared among the related information catalogs. If the value of *type* already exists, it is used as a search argument.

The maximum length for the value is 8 characters. The value is stored in uppercase characters. This value can start with the characters A - Z, @, #, or $, and can contain any of these characters plus 0 - 9 and _. No leading blanks or embedded blanks are allowed.

After you create the object type, you cannot change the value of *type*.

CATEGORY
Specifies which category this object type belongs to.

Required keyword.

category
Specifies an Information Catalog Manager object category. This value can be one of the following:
GROUPING
ELEMENTAL
SUPPORT
CONTACT
DICTIONARY

You cannot specify PROGRAM or ATTACHMENT as the category for a new object type.

You cannot change the information on this keyword after the object type is defined.

EXTNAME
Specifies a longer, descriptive name for the object type. Required keyword.

ext_name
Specifies an extended, descriptive name for the object type. The maximum length for *ext_name* is 80 characters.

This name must be unique within related information catalogs.

The value of *ext_name* is stored in mixed case.

You can change the information on this keyword after the object type is defined.

PHYNAME
Specifies the name to use when creating the database table that contains information about this object type.

Optional keyword.

table_name

Specifies the name to use when creating the database table that contains object type information.

The maximum length of the name is defined when the Information Catalog Manager is installed. The *table_name* value must be unique within the information catalog and cannot contain any SQL reserved words.

By default, *table_name* is the *type* that is specified for the **TYPE** keyword. This value is not case sensitive; you can specify this value with uppercase or lowercase characters.

This value can start with the characters A - Z, @, #, or $, and can contain any of these characters, plus 0 - 9 and _. No leading blanks or embedded blanks are allowed. This value cannot be any of the SQL reserved words for the database that is used for the information catalog.

After the table is created, you cannot change its name.

ICWFILE

Specifies the file that contains the Windows icon that is associated with the object type.

Optional keyword.

Windows_ICON_file_name

Specifies the name of the Windows icon file to associate with the object type. The maximum length of *Windows_ICON_file_name* is 254 characters. However, this name, combined with the icon path (ICOPATH), can have a maximum length of 259, so the true maximum length depends on the length of the icon path. This file can have any extension. This value is not case sensitive; you can specify this value by using uppercase or lowercase characters.

You cannot specify the drive and path information that identifies where the icon file resides using this keyword. You must specify this information as an input parameter for the FLGImport API call (see the *Information Catalog Manager Programming Guide and Reference*), the import function on the user interface (see “Importing a tag language file from the command line” on page 42), or the IMPORT option of the DGUIDE command (see “Importing a tag language file from the command line” on page 42).

You can change this value after the object type is created by using ACTION.OBJTYPE(UPDATE). After you specify an icon file to associate with an object type, you can change the associated icon, but the object type must always be associated with an icon.

ACTION.OBJTYPE(APPEND)

Context:

```
:ACTION.OBJTYPE(APPEND)
:OBJECT.TYPE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 61. Using the OBJECT tag when adding properties to object types

Keywords:

TYPE
Specifies the name (*type*) of an object type.

Required keyword.

type
Identifies a specific object type by its 8-character short name.

ACTION.OBJTYPE(DELETE) or ACTION.OBJTYPE(DELETE_EXT)
Deleting an existing object type.

Context:

```
:ACTION.OBJTYPE(DELETE)
:OBJECT.TYPE()
```

Figure 62. Using the OBJECT tag when deleting object types

```
:ACTION.OBJTYPE(DELETE_EXT)
:OBJECT.TYPE()
```

Figure 63. Using the OBJECT tag when deleting object types and all objects of that type

Keywords:

TYPE
Specifies the name (*type*) of an object type.

Required keyword.

type
Identifies a specific object type by its 8-character short name.

ACTION.OBJTYPE(UPDATE)
Updating object type information.

Context:

```
:ACTION.OBJTYPE(UPDATE)
:OBJECT.TYPE() EXTNAME() ICOFILE() ICWFILE()
```

Figure 64. Using the OBJECT tag when updating object types

Keywords:

TYPE
Specifies the name (*type*) of an object type.

Required keyword.

type
Identifies a specific object type by its 8-character short name. You cannot update this value.

EXTNAME
Specifies a descriptive name for the object type. Optional keyword.

ext_name
Specifies an extended, descriptive name for the object type. The maximum length for *ext_name* is 80 characters.

You can update this value.

This name must be unique within related information catalogs.

The value of *ext_name* is stored in mixed case.

ICWFILE
Specifies the file that contains the Windows icon that is associated with the object type.

Optional keyword.

Windows_ICON_file_name
Specifies the name of the Windows icon file to associate with the object type.

You can update this value.

The maximum length of *Windows_ICON_file_name* is 254 characters. You cannot use this keyword to specify the drive and path information that identifies where the ICON file resides. You must specify this information as an input parameter for the FLGImport API call, the import function on the user interface, or the IMPORT option of the Information Catalog Manager command.

ACTION.OBJINST
Adding, updating, deleting, or merging objects

Context:

```
:ACTION.OBJINST(ADD)
:OBJECT.TYPE()
:INSTANCE.short_name()
```

Figure 65. Using the OBJECT tag when adding objects

```
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE()
:INSTANCE.short_name()
```

Figure 66. Using the OBJECT tag when merging objects

```
:ACTION.OBJINST(UPDATE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) short_name()
```

Figure 67. Using the OBJECT tag when updating objects

```
:ACTION.OBJINST(DELETE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 68. Using the OBJECT tag when deleting objects

Keywords:

TYPE
Specifies the name (*type*) of an object type.

Required keyword.

type
Identifies a specific object type by its 8-character short name.

PROPERTY

Defines a property that belongs to an object type.

This tag is required following these ACTION tags:
:ACTION.OBJTYPE(ADD)
:ACTION.OBJTYPE(MERGE)
:ACTION.OBJTYPE(APPEND)

Syntax

```
:PROPERTY.EXTNAME(ext_name) DT(data_type) DL(data_length)
            SHRTNAME(short_name) NULLS(Y | N) UUISEQ(UUI_number)
```

Context

```
:ACTION.OBJTYPE(ADD)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 69. Using the PROPERTY tag when adding object types

```
:ACTION.OBJTYPE(MERGE)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 70. Using the PROPERTY tag when merging object types

```
:ACTION.OBJTYPE(APPEND)
:OBJECT.TYPE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 71. Using the PROPERTY tag when adding properties to object types

Keywords

EXTNAME
Specifies a descriptive name for the property.

Required keyword.

ext_name
Specifies an extended descriptive name.

The maximum length of *ext_name* is 80 characters. The *ext_name* must be unique within the object type. *ext_name* is stored in mixed case.

DT
Specifies the data type for the property.

Required keyword.

data_type
The data type for the property. You can specify this value in either uppercase or lowercase. Valid values are:

C	Character
V	Variable character

L	Long variable character
T	Timestamp

DL
Specifies the data length or maximum data length for the property.

Required property.

data_length
The data length or maximum data length for the property. Valid values for *data_length* depend on the *data_type* that is defined for this property:

data_type	**Maximum value for data_length**
C (character)	Maximum length is 254
V (variable character)	Maximum length is 4000
L (long variable character)	Maximum length is 32700
T (timestamp)	Always 26 characters

SHRTNAME
Specifies the property short name.

Required keyword.

short_name
The short name for the property. The *short_name* value can be up to 8 characters long. This value can contain only SBCS characters.

This value is stored as uppercase characters; any lowercase characters are converted to uppercase.

This value can start with the characters A - Z, @, #, or $, and can contain any of these characters, plus 0 - 9 and _. No leading blanks or embedded blanks are allowed.

This value cannot be any of the SQL reserved words for the database that is used for the information catalog. Do not specify the property short names of the following required properties for every Information Catalog Manager object type: OBJTYPID, INSTIDNT, UPDATIME, or UPDATEBY.

NULLS
Specifies whether a value for the property is required for every object. This value can be specified in uppercase or lowercase.

Required keyword.

Y indicates that this value can be null. When appending a new property with the ACTION.OBJTYPE(APPEND) tag, you must specify NULLS(Y), because appended properties must be optional.

N indicates that a value for this property is required. If no data exists for a required property when an object is added to the information catalog, a not-applicable symbol is entered for the required value for data types of CHAR, VARCHAR, and LONG VARCHAR. For a required value with a data type of TIMESTAMP, the following value is entered: 9999-12-31-24.00.00.000000

UUISEQ
Identifies the properties that are used in the UUI.

Optional keyword; the default value is 0. The UUISEQ keyword is optional for properties that are not part of the UUI. The UUI is a set of properties that are defined by the administrator as the key that uniquely identifies each object.

UUI_number
Specifies the position of the property in the UUI sequence. Valid values are 0, 1, 2, 3, 4, and 5. The value 0 means that the property is not part of the UUI. A nonzero value for *UUI_number* indicates that the property is part of the UUI.

All object types defined in the tag language file must have at least one property that is part of the UUI. The UUI can consist of up to 5 properties.

At least one property must be defined as part of the UUI.

When assigning *UUI_number* values to more than one property, the numbers of the UUI properties must range from 1 to the number of properties in the UUI. For example, if three properties are defined as part of the UUI, the *UUI_number* values must be 1, 2, and 3. You cannot skip numbers in the sequence. The *UUI_number* values do not need to be in the same order that the properties are specified.

Rules

- You can define the reserved property NAME as part of the UUI when you add a new object type or merge object types. Figure 72 shows the general syntax for identifying NAME as a UUI property.

```
:ACTION.OBJTYPE(ADD)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.SHRTNAME(NAME) UUISEQ()
```

Figure 72. Example of specifying the NAME property as part of the UUI

  Empty parentheses in this figure denote values that you must provide in a tag language file.
- The maximum length of the UUI fields is 254 bytes.

RELTYPE

Identifies the type of relationship that to add or delete and the object types of the objects involved in the relationship.

This tag is required immediately following these tags:
:ACTION.RELATION(ADD)
:ACTION.RELATION(DELETE)

Syntax

```
:RELTYPE.TYPE(CONTAIN | CONTACT | ATTACHMENT | LINK)
                    SOURCETYPE(source_type) TARGETTYPE(target_type)
```

Context

```
:ACTION.RELATION(ADD)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 73. Using the RELTYPE tag when adding relationships

```
:ACTION.RELATION(DELETE)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 74. Using the RELTYPE tag when deleting relationships

Keywords

TYPE
Specifies the type of relationship.

Required keyword.

Valid values are:

ATTACHMENT
Attachment relationship: target object is attached to the source object.

CONTACT
Contact relationship: Source object is associated with the target Contact object.

CONTAIN
Contains relationship: source object contains the target object.

LINK Link relationship: source object is linked with the target object.

SOURCETYPE
Identifies the source object type.

Required keyword.

source_type
The source object type name *source_type* corresponds to the *type* value for the TYPE keyword of the OBJECT tag. The maximum length for *source_type* is 8 characters. This value is not case sensitive; you can specify this value with uppercase or lowercase characters.

For an Attachment relationship, *source_type* is a non-Attachment object-type name.

For a Contains relationship, *source_type* is the container object type name.

For a Contact or link relationship *source_type* is the Grouping or Elemental object type name.

TARGETYPE
Identifies the target object type.

Required keyword.

target_type
The target object type name. *target_type* corresponds to the *type* value for the TYPE keyword on the OBJECT tag. The maximum length for *target_type* is 8 characters. This value is not case sensitive; you can specify this value with uppercase or lowercase characters.

For an Attachment relationship, *target_type* is the Attachment object-type name.

For a Contains relationship, *target_type* is the contained object type name.

For a Contact relationship, *target_type* is the Contact object-type name.

For a link relationship, *target_type* is a Grouping or Elemental object type name.

TAB

Specifies a tab within a property value.

The Information Catalog Manager reads only TAB tags that are specified within non-UUI property values and ignores all others.

Syntax

```
:TAB.
```

Rules

Use TAB tags only within the specification of *property_values* in INSTANCE tags.

Chapter 10. What a tag language file should look like

You can use the tags to add, delete, and update object types and objects. Information Catalog Manager tags are contextual; you specify tags in different combinations depending on what you want to do.

Start your tag language file with DISKCNTL

Start the tag language file with a DISKCNTL tag if the file is on a removable disk, such as a diskette. For example:

```
:DISKCNTL.SEQUENCE(01,+)
```

If the tag language file is on more than one diskette, then DISKCNTL must be the first tag in each section of the tag language file on each diskette. If the tag language file is on a fixed disk, then DISKCNTL is ignored.

Define your additions, changes, and deletions

You use the tag language to define actions and the objects of those actions.

Defining what you want to do

The ACTION tag tells Information Catalog Manager what you want to do. The keyword tells the Information Catalog Manager what kind of information you want to maintain. The option tells the Information Catalog Manager what task you want to perform.

:ACTION.OBJINST(*option*)
Maintaining objects.

:ACTION.OBJTYPE(*option*)
Maintaining object types.

:ACTION.RELTYPE(*option*)
Maintaining object relationships.

Defining the information

After you have specified what you want to do, you need to define precisely what information you are adding, changing, or deleting.

To define:	**Use these tags:**
Existing object type	OBJECT
Object type to be merged	OBJECT and PROPERTY
New object type	OBJECT and PROPERTY
New properties for an object type	OBJECT and PROPERTY

To define:	**Use these tags:**
New or existing object	OBJECT and INSTANCE
New or existing object relationship	RELTYPE and INSTANCE

Putting it all together

The keywords and values that are required for OBJECT, INSTANCE, and PROPERTY tags are different depending on what they are identifying to add, change, or delete. The sequence of tags within each ACTION tag is:

:ACTION.OBJINST(*option***)**

```
:ACTION.OBJINST(ADD)
:OBJECT.TYPE()
:INSTANCE.short_name() ...
```

```
:ACTION.OBJINST(DELETE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

```
:ACTION.OBJINST(DELETE_TREE_ALL)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

```
:ACTION.OBJINST(DELETE_TREE_REL)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

```
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE()
:INSTANCE.short_name() ...
```

```
:ACTION.OBJINST(UPDATE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) short_name()
```

:ACTION.OBJTYPE(*option***)**

```
:ACTION.OBJTYPE(ADD)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

```
:ACTION.OBJTYPE(APPEND)
:OBJECT.TYPE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

```
:ACTION.OBJTYPE(DELETE)
:OBJECT.TYPE()
```

```
:ACTION.OBJTYPE(DELETE_EXT)
:OBJECT.TYPE()
```

```
:ACTION.OBJTYPE(MERGE)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

```
:ACTION.OBJTYPE(UPDATE)
:OBJECT.TYPE() EXTNAME() ICOFILE() ICWFILE()
```

:ACTION.RELATION(*option***)**

```
:ACTION.RELATION(ADD)
:RELTYPE.TYPE(CONTAIN | CONTACT | ATTACHMENT | LINK) SOURCETYPE(type)
TARGETYPE(type)
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
:ACTION.RELATION(DELETE)
:RELTYPE.TYPE(CONTAIN | CONTACT | ATTACHMENT | LINK) SOURCETYPE(type)
TARGETYPE(type)
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

For specific information about the format of the INSTANCE, OBJECT, and PROPERTY tags, see “INSTANCE” on page 231, “OBJECT” on page 237, or “PROPERTY” on page 242.

Committing changes to the database

The COMMIT tag commits changes to the information catalog database. When a COMMIT tag processes, the echo file is emptied before the next set of tags starts processing. This ensures that the echo file contains only tags that describe uncommitted changes.

If the Information Catalog Manager encounters an error, it rolls back the database to the last committed checkpoint. Insert COMMIT tags in your file to keep your data consistent, and to limit the number of changes that are canceled when the database is rolled back.

You can insert a COMMIT tag after any complete set of tags that define an action. Do not insert a COMMIT tag between the ACTION tag and the last tag that defines the data that is associated with the ACTION tag.

```
:COMMIT.CHKPT(20)
```

Putting comments in the tag language file

You can use the COMMENT tag to put information in the tag language file, such as notes and labels, that you do not want to import into your information catalog.

```
:COMMENT.Updating the LASTDATE property
```

Part 3. Supplied program and macro reference

Chapter 11. Supplied Data Warehouse Center programs

The Data Warehouse Center supplies the following programs to support integration with the Data Warehouse Center:

- VWPEXUNX
- ISV_Sample

VWPEXUNX

The VWPEXUNX program remotely issues a command or runs a program. VWPEXUNX runs on Windows NT, Windows 2000, and UNIX®.

If you are running the VWPEXUNX program on Windows NT or Windows 2000, the REXECD program must also be running on the workstation.

Parameters

Table 121 shows the parameter list for the VWPEXUNX program. The list includes the predefined token for a parameter if one exists.

Table 121. Parameters for VWPEXUNX

Order	Description
1	The remote host name.
2	The remote user ID.
3	The remote program to execute.
4	The remote error file.
5	The remote warning file. If there is no warning file, specify - (the not-applicable symbol).
6	The remote log (summary) file. If there is no log file, specify - (the not-applicable symbol).
7	The remote operating system type. Specify either UNIX, WINNT, or WIN2000.
8	The password type. Specify either PasswordNotRequired, EnterPassword, or GetPassword.

Table 121. Parameters for VWPEXUNX (continued)

Order	Description
9	The password value if the password type is EnterPassword. - (not-applicable symbol) if the password type is PasswordNot Required. The password program if password type is GetPassword. The password program must reside on the agent site that is selected for the step. The program must write a file that contains the password to use in the first line of the file. It must return 0 if it runs correctly.
10	The password program parameters if the password type is GetPassword

The following example shows how to start the VWPEXUNX program from a command prompt. The command must be typed all on one line. The line break shown in this example is not significant.

```
vwpexunx tomari labriejj db2cmd \usr\labriejj\db2cmd.err - -
UNIX EnterPassword mypass
```

tomari	The name of the remote host
labriejj	The user ID used to access the remote host
db2cmd	The remote program to run
\usr\labriejj\db2cmd.err	The path and name of the remote error file
-	No remote warning file exists
-	No remote log (summary) file exists
UNIX	The remote operating system
EnterPassword	The password type
mypass	The password

Return codes

The VWPEXUNX program uses the remote error file to determine the success or failure of the remote command or program:

- If the error file is empty or nonexistent, the VWPEXUNX program returns an error code that indicates success.
- If the error file is not empty, the VWPEXUNX program:
  - Saves the contents of the error file in a temporary file.
  - Returns an error code that indicates failure.

The VWPEXUNX program does not check the contents of the remote error file.

Table 122 lists the return codes for the VWPEXUNX program.

Table 122. Return codes for the VWPEXUNX program

Return code	Description
0	The program ran successfully.
4	The program ran with a warning. The program could not erase the password file after the password program ran.
8	Parameter error. Too few or too many parameters were supplied to the program, or an invalid value was supplied for a parameter.
16	Internal error. The program detected an internal error, such as the inability to open, create, or write to a temporary file.
48	Environment variable error. The *VWS_LOGGING* environment variable was not set.
52	Get password program error. The program detected a password program error, such as a missing program, an invalid name, or the wrong number of parameters
56	Remote execution error. The program detected a remote execution error, such as the following errors: • An incorrect user ID or password was supplied. • A remote file was not found. • A remote host is not responding. • The supplied user ID is not authorized to create or read the remote file.

Log files

The VWPEXUNX program writes a trace file to the directory that the *VWS_LOGGING* environment variable specifies.

ISV_Sample

The ISV_Sample program reads metadata from ODBC data sources and generates Data Warehouse Center objects from the metadata. The ISV_Sample program runs on Windows NT and Windows 2000.

Table 123 shows the parameter list for the ISV_Sample program.

No predefined tokens exist for the parameters.

Table 123. Parameters for ISV_Sample

Order	Description
1	ODBC DSN from which to extract metadata
2	ODBC user ID
3	ODBC password

The following example shows how to start the ISV_Sample program:

```
ISV_Sample SAMPLE labriejj mypass
```

SAMPLE	The ODBC DSN from which to read metadata
labriejj	The user ID used to access the ODBC DSN
mypass	The password used to access the ODBC DSN

The ISV_Sample program uses the ISV_VWP program. Steps call the ISV_VWP program to write the input parameters to an output file.

Chapter 12. Net.Data® macros

The Information Catalog Manager for the Web uses Net.Data® macros to display data on the Web and search for data in a database. If you are familiar with Net.Data and its macros, you can customize these macros to meet the requirements of your organization.

For example, the Information Catalog Manager for the Web requires a user ID and password by default. You can customize the macros to call your own security program instead.

This chapter lists the files that are included with the Information Catalog Manager for the Web. For more information about Net.Data and its macros, see the *Net.Data Programming Guide* and *Net.Data Reference Guide*.

Information Catalog Manager for the Web files

To work with Information Catalog Manager for the Web files, you must first perform a custom installation of the Administration Client and select Information Catalog Manager for the Web. The files are installed in the `x:\sqllib\icuweb` directory.

The file names are lowercase to follow the AIX® naming convention.

Table 124 lists the Information Catalog Manager for the Web files that contain Net.Data macros, which are located in the `x:\sqllib\icuweb\macro` directory.

Table 124. Information Catalog Manager Web Net.Data macros

File name	Description
dg_list.mac	Displays the results of a search, tree, or subject call
dg_desc.mac	Displays the results of a description view
dg_frame.mac	Creates the three-frame page
dg_advsearch.mac	Performs an advanced search
dg_comment.mac	Creates or updates a comment
dg_home.mac	Displays the Information Catalog Manager home page
dg_tableviewer.mac	Displays sample data

Information Catalog Manager for the Web files

Table 125 lists the Information Catalog Manager for the Web files that contain Net.Data include files, which are located in the `x:\sqllib\icuweb\macro` directory.

Table 125. Net.Data include files

File name	Description
dg_desc.hti	Include file with common functions for description view
dg_home.hti	Include file with a list of information catalogs to display on the Information Catalog Manager home page
dg_strings.hti	Include file with translatable strings
dg_config.hti	Include file with installation configurable variables
dg_graphics.hti	Include file with graphics look and feel definitions

Table 126 displays the Information Catalog Manager for the Web files that contain HTML, which are located in the `x:\sqllib\icuweb\html` directory.

Table 126. Information Catalog Manager for the Web HTML files

File Name	Description
*.htm	Help files

Table 127 lists the Information Catalog Manager for the Web graphic files, which are located in the `x:\sqllib\icuweb\icons` directory.

In addition to the graphics files listed below, you can also create unique icons for any new object type that you create in the Information Catalog Manager. For more information on creating object type icons, see the *Information Catalog Manager Administration Guide.*

Table 127. Information Catalog Manager for the Web graphics files

File name	Description
dg_ibmlogo.gif	IBM logo
dg_lgudblogo.gif	Large DB2 logo on Home
dg_smudblogo.gif	Small DB2 logo on header
dg_curve.gif	Small curve joining header and menu
dg_lgappldata.gif	Large Application Data
dg_smappldata.gif	Small Application Data

Table 127. Information Catalog Manager for the Web graphics files (continued)

File name	Description
dg_lgapproach.gif	Large Lotus Approach
dg_smapproach.gif	Small Lotus Approach
dg_lgaudio.gif	Large Audio Clips
dg_smaudio.gif	Small Audio Clips
dg_lgcharts.gif	Large Charts
dg_smcharts.gif	Small Charts
dg_lgcolumns.gif	Large Columns
dg_smcolumns.gif	Small Columns
dg_lgcomments.gif	Large Comments
dg_smcomments.gif	Small Comments
dg_lgcontact.gif	Large People to contact
dg_smcontact.gif	Small People to contact
dg_lgdatabas.gif	Large Databases
dg_smdatabas.gif	Small Databases
dg_lgimsdbd.gif	Large IMS database definitions (DBD)
dg_smimsdbd.gif	Small IMS database definitions (DBD)
dg_lgdgnews.gif	Large News
dg_smdgnews.gif	Small News
dg_lgdimenson.gif	Large Dimensions within a multi-dimensional database
dg_smdimenson.gif	Small Dimensions within a multi-dimensional database
dg_lgdocs.gif	Large Documents
dg_smdocs.gif	Small Documents
dg_lgelement.gif	Large Elements
dg_smelement.gif	Small Elements
dg_lgfile.gif	Large Files
dg_smfile.gif	Small Files
dg_lgfilter.gif	Large Transformations
dg_smfilter.gif	Small Transformations
dg_lgglossary.gif	Large Glossary entries
dg_smglossary.gif	Small Glossary entries
dg_lgimages.gif	Large Images or graphics

Table 127. Information Catalog Manager for the Web graphics files (continued)

File name	Description
dg_smimages.gif	Small Images or graphics
dg_lginfogrps.gif	Large Business subject areas
dg_sminfogrps.gif	Small Business subject areas
dg_lginternet.gif	Large Internet documents
dg_sminternet.gif	Small Internet documents
dg_lgmember.gif	Large Members within a multidimensional database
dg_smmember.gif	Small ″Members within a multidimensional database
dg_lgolapmodl.gif	Large Multidimensional database
dg_smolapmodl.gif	Small Multidimensional database
dg_lgolnews.gif	Large Online news services
dg_smolnews.gif	Small Online news services
dg_lgolpubs.gif	Large Online news services
dg_smolpubs.gif	Small Online news services
dg_lgiimspcb.gif	Large IMS program control block (PCB)
dg_smimspcb.gif	Small IMS program control block (PCB)
dg_lgpresent.gif	Large Presentations
dg_smpresent.gif	Small Presentations
dg_lgimspsb.gif	Large IMS program specifications (PSB)
dg_smimspsb.gif	Small IMS program specifications (PSB)
dg_lgrecord.gif	Large Records
dg_smrecord.gif	Small Records
dg_lgreports.gif	Large Text-based reports
dg_smreports.gif	Small Text-based reports
dg_lgmsseg.gif	Large IMS segment
dg_smimsseg.gif	Small IMS segment
dg_lgssheets.gif	Large Spreadsheet
dg_smssheets.gif	Small Spreadsheet
dg_lgsubschem.gif	Large Subschemas
dg_smsubschem.gif	Small Subschemas
dg_lgtables.gif	Large Relational tables and views
dg_smtables.gif	Small Relational tables and views

Table 127. Information Catalog Manager for the Web graphics files (continued)

File name	Description
dg_lgvideo.gif	Large Video clips
dg_smvideo.gif	Small Video clips
dg_lggrouping.gif	Large Grouping - default category icon
dg_smgrouping.gif	Small Grouping- default category icon
dg_lgelemental.gif	Large Elemental- default category icon
dg_smelemental.gif	Small Elemental- default category icon
dg_lgcontact.gif	Large Contact- default category icon
dg_smcontact.gif	Small Contact- default category icon
dg_lgdictionary.gif	Large Dictionary- default category icon
dg_smdictionary.gif	Small Dictionary- default category icon
dg_lgsupport.gif	Large Support- default category icon
dg_smsupport.gif	Small Support- default category icon
dg_lgattachment.gif	Large Attachment- default category icon
dg_smattachment.gif	Small Attachment- default category icon
dg_collapse.gif	tree - collapse icon
dg_expand.gif	tree - expand icon
dg_lmore.gif	description - long property (more arrow)
dg_clear.gif	clear graphic for spacing

Part 4. Appendixes

Appendix A. Template planning worksheet

Use this worksheet to collect the values that your partner application needs to provide.

Write the value of the token in the table. For tokens that have a specific list of allowed values, circle one of the allowed values.

Table 128. Tokens for required metadata in the templates

Token	Value
**AgentSite*	
**AgentSiteContact*	
**AgentSiteDescription*	
**AgentSiteNotes*	
**AgentSiteOSType*	One of the following values: **ISV_windowsNT** Windows NT **ISV_AIX** AIX **ISV_os2** OS/2 **ISV_as400** AS/400 **ISV_Solaris** SUN **ISV_MVS** MVS
**AgentSiteTCPIPHostName*	
**AgentSiteUserid*	
**ColumnAllowsNulls*	One of the following values: **ISV_NULLSYES** The column allows null data. **ISV_NULLSNO** The column does not allow null data.

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**ColumnDataIsText*	One of the following values: **ISV_ISTEXTYES** The column contains only text data. **ISV_ISTEXTNO** The column does not contain only text data.
**ColumnDescription*	
**ColumnEditionType*	One of the following values: **ISV_ColumnIsEditionColumn** The column is an edition column. **ISV_ColumnIsNormal** The column is a normal column.
**ColumnKeyPosition*	
**ColumnLength*	
**ColumnName*	

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**ColumnNativeDataType*	One of the following values: ISV_NATIVE_CHAR ISV_NATIVE_VARCHAR ISV_NATIVE_LONGVARCHAR ISV_NATIVE_VARCHAR2 ISV_NATIVE_GRAPHIC ISV_NATIVE_VARGRAPHIC ISV_NATIVE_LONGVARGRAPHIC ISV_NATIVE_CLOB ISV_NATIVE_INT ISV_NATIVE_TINYINT ISV_NATIVE_BLOB ISV_NATIVE_SMALLINT ISV_NATIVE_INTEGER ISV_NATIVE_FLOAT ISV_NATIVE_SMALLFLOAT ISV_NATIVE_DOUBLE ISV_NATIVE_REAL ISV_NATIVE_DECIMAL ISV_NATIVE_SMALLMONEY ISV_NATIVE_MONEY ISV_NATIVE_NUMBER ISV_NATIVE_NUMERIC ISV_NATIVE_DATE ISV_NATIVE_TIME ISV_NATIVE_TIMESTAMP ISV_NATIVE_LONG ISV_NATIVE_RAW ISV_NATIVE_LONGRAW ISV_NATIVE_DATETIME ISV_NATIVE_SMALLDATETIME ISV_NATIVE_SYSNAME ISV_NATIVE_TEXT ISV_NATIVE_BINARY

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**ColumnNativeDataType* (continued)	One of the following values: ISV_NATIVE_VARBINARY ISV_NATIVE_LONGVARBINARY ISV_NATIVE_BIT ISV_NATIVE_IMAGE ISV_NATIVE_SERIAL ISV_NATIVE_DATETIMEYEARTOFRACTION ISV_NATIVE_DBCLOB ISV_NATIVE_BIGINT
**ColumnNotes*	
**ColumnOffsetFromZero*	
**ColumnOrdinalNumber*	
**ColumnPositionNumber*	
**ColumnPrecision*	
**ColumnUserActions*	
**CurrentCheckPointID++*	
**DatabaseContact*	
**DatabaseDescription*	
**DatabaseName*	
**DatabaseNotes*	
**DatabasePhysicalName*	

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**DatabaseType*	One of the following values: **ISV_IR_DB2Family** DB2 Family **ISV_IR_Oracle** Oracle **ISV_IR_Sybase** Sybase **ISV_IR_MSSQLServer** Microsoft SQLServer **ISV_IR_Informix** Informix **ISV_IR_GenericODBC** Generic ODBC **ISV_IR_FFLan** Flat File LAN **ISV_IR_VSAM** VSAM **ISV_IR_IMS** IMS
**DatabaseTypeExtended*	One of the following values: **ISV_IR_DB2400CISC** DB2 UDB for AS/400® for CISC **ISV_IR_DB2400RISC** DB2 UDB for AS/400 for RISC **ISV_IR_FFLanLocalCmd** Local flat file **ISV_IR_FFLanFTPCopy** Local flat file sent using FTP from a remote system
**DatabaseServerName*	
**DatabaseUserid*	
**DatabaseVersion*	
**PostStepName*	
**ProcessContact*	

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**ProcessDescription*	
**ProcessName*	
**ProcessNotes*	
**ProcessType*	One of the following values: **ISV_ProcessType_Normal** Process is a normal user process. **ISV_ProcessType_Meta_pub** Process is a metadata publication process. **ISV_ProcessType_Notify** Process is a notification process.
**SecurityGroup*	`ISV_DEFAULTSECURITYGROUP`
**StarSchemaContact*	
**StarSchemaDBName*	
**StarSchemaDescription*	
**StarSchemaName*	
**StarSchemaNotes*	
**StepCommit*	One of the following values: **ISV_Step_Incremental_Commit_On** The data is to be incrementaly commited at the target. **ISV_Step_Incremental_Commit_Off** The data is not to be incrementaly commited at the target.
**StepCommitAfterNumberRows*	
**StepContact*	

Table 128. Tokens for required metadata in the templates (continued)

Token	**Value**
**StepDataNotPresent*	One of the following values: **ISV_StepDataNotPresent_OK** If data is not present, continue processing. **ISV_StepDataNotPresent_Warning** If data is not present, issue a warning and continue processing. **ISV_StepDataNotPresent_Error** If data is not present, issue an error message and stop processing.
**StepDescription*	
**StepExternalPopulation*	One of the following values: **ISV_StepExternalNo** The table will not be externally populated by other means. **ISV_StepExternalYes** The table will be externally populated by other means.
**StepName*	
**StepNotes*	
**StepSelectStatement*	
**StepSelectStatementGenerated*	One of the following values: **ISV_StepSelectStatementNo** The SELECT statement is not generated, but is included in the **StepSelectStatement*. **ISV_StepSelectStatementYes** The SELECT statement is generated, and **StepSelectStatement* is ignored.

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**StepSQLWarning*	One of the following values: **ISV_StepSQLWarning_OK** If an SQL warning occurs, continue processing. **ISV_StepSQLWarning_Warning** If an SQL warning occurs, issue a warning and continue processing. **ISV_StepSQLWarning_Error** If an SQL warning occurs, issue an error and stop processing.
**StepType*	One of the following values: **ISV_StepType_Editioned_Append** The data in the table will be appended when the Step is run. **ISV_StepType_Full_Replace** The data in the table will be replaced when the Step is run. **ISV_StepType_Uneditioned_Append** The data in the table will be appended when the Step is run. **ISV_StepType_VWP_Population** The data in the table is populated by a Data Warehouse Center program.
**SubjectArea*	
**SubjectAreaContact*	
**SubjectAreaDescription*	
**SubjectAreaNotes*	
**TableBinaryIfFile*	One of the following values: **ISV_DR_FILE_IS_BINARY** The file is binary. **ISV_DR_FILE_IS_NOT_BINARY** The file is in ASCII or mixed format.

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**TableCreatedByDWC*	One of the following values: **ISV_TableIsToBeCreatedByDWC** The table is to be created by the Data Warehouse Center. **ISV_TableIsNotToBeCreatedByDWC** The table is not to be created by the Data Warehouse Center.
**TableCreateStatement*	
**TableDelimiterIfFile*	
**TableDescription*	
**TableFirstRowNamesIfFile*	One of the following values: **ISV_DR_ROW_CONTAINS_NAMES** The first row of the file contains column names. **ISV_DR_ROW_DOES_NOT_CONTAIN_NAMES** The first row of the file contains data.
**TableFullName*	
**TableGenerateCreateStatement*	One of the following values: **ISV_GenerateCreateTableStmt** The Data Warehouse Center should generate the CREATE TABLE statement. **ISV_DoNotGenerateCreateTableStmt** The Data Warehouse Center should not generate the CREATE TABLE statement.
**TableGrantedToPublic*	One of the following values: **ISV_GrantTableAccessToPublic** Grant PUBLIC access to this table. **ISV_DoNotGrantTableAccessToPublic** Do not grant PUBLIC access to this table.

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**TableIsAnAlias*	One of the following values: **ISV_TableIsAnAlias** This table is an alias for another table. **ISV_TableIsNotAnAlias** This table is not an alias for another table.
**TableIsADimensionTable*	One of the following values: **ISV_TableIsADimensionalTable** The table is a dimensional table. **ISV_TableIsNotADimensionalTable** The table is not a dimensional table.
**TableIsAFactTable*	One of the following values: **ISV_TableIsAFactTable** The table is a fact table. **ISV_TableIsNotAFactTable** The table is not a fact table.
**TableIsAView*	One of the following values: **ISV_TableIsAView** The table is a view. **ISV_TableIsNotAView** The table is not a view.
**TableIsPersistent*	One of the following values: **ISV_TableIsPersistent** The table is to be considered persistent. **ISV_TableIsTransient** The table is to be considered transient.
**TableMaximumEditions*	
**TableNotes*	
**TableOwner*	
**TablePhysicalName*	

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**TableTypeIfFile*	One of the following values: **ISV_DR_REL_TABLE** The table is a relational table. **ISV_DR_COMMA_DELIMITED** The columns in the file are separated by commas. **ISV_DR_FIXED_FORMAT** The columns in the file are in fixed format. **ISV_DR_TAB_DELIMITED** The columns in the file are separated by tabs. **ISV_DR_CHAR_DELIMITED** The columns in the file are separated by the value of **TableDelimiterIfFile*.
**VWPGroup*	
**VWPGroupDescription*	
**VWPGroupNotes*	
**VWPProgramInstanceKey*	
**VWPProgramInstanceParameterData*	
**VWPProgramInstanceParameterKey*	
**VWPProgramInstanceParameterName*	
**VWPProgramInstanceParameterOrder*	
**VWPProgramInstanceParameterType*	One of the following values: **ISV_ParameterTypeNone** The parameter type is unknown. **ISV_ParameterTypeCharacter** The parameter type is character. **ISV_ParameterTypeNumeric** The parameter type is numeric. **ISV_ParameterTypePassword** The parameter type is password.
**VWPProgramTemplateDescription*	
**VWPProgramTemplateExecutableName*	

Table 128. Tokens for required metadata in the templates (continued)

Token	Value
**VWPProgramTemplateFunctionName*	
**VWPProgramTemplateName*	
**VWPProgramTemplateNotes*	
**VWPProgramTemplateType*	One of the following values: **ISV_PROGRAMTYPEDLL** The Data Warehouse Center program is loaded from a dynamic link library (DLL) or is a load module. **ISV_PROGRAMTYPECOMMAND** The Data Warehouse Center program is a command file. **ISV_PROGRAMTYPEEXECUTABLE** The Data Warehouse Center program is an executable file.
**VWPProgramTemplateParameterData*	
**VWPProgramTemplateParameterKey*	
**VWPProgramTemplateParameterName*	
**VWPProgramTemplateParameterOrder*	
**VWPProgramTemplateParameterType*	One of the following values: **ISV_ParameterTypeNone** The parameter type is unknown. **ISV_ParameterTypeCharacter** The parameter type is character. **ISV_ParameterTypeNumeric** The parameter type is numeric. **ISV_ParameterTypePassword** The parameter type is password.

Appendix B. Templates supported by Visual Warehouse 5.2

Refer to this chapter for detailed information about templates that are offered with and supported by Version 5.2 of Visual Warehouse and DataGuide®. The section for each template lists the tokens for the template. It provides the allowed values and lengths of values for each token.

Note: The templates described in this chapter have been deprecated and will not be enhanced.

If your interchange program does not have a value for a token, it should set the token to `ISV_DEFAULTVALUE`. However, you must specify a value other than `ISV_DEFAULTVALUE` for any token that is required.

Because there is no template for security groups, your program must specify the value `ISV_DEFAULTSECURITYGROUP` for any instances of the **SecurityGroup* token.

If the template does not set a Visual Warehouse parameter, the Visual Warehouse definition will have the default value of the parameter. For example, Visual Warehouse sets the Retry Count and Retry Interval parameters for source databases to their default values.

Table 129 lists the metadata templates that are supplied with Visual Warehouse and the section that covers each template.

Table 129. Metadata templates supported by Visual Warehouse 5.2

BusinessView.tag	Defines a business view that is to be managed by Visual Warehouse.	"BusinessView.tag" on page 280
BusinessViewInputTable.tag	Specifies that a business view uses a given source table.	"BusinessViewInputTable.tag" on page 285
BusinessViewOutputTable.tag	Specifies that a business view uses a given target table.	"BusinessViewOutputTable.tag" on page 286
BusinessViewVWPOutputTable.tag	Specifies a relationship between a business view that uses a Visual Warehouse program and the output table for the Visual Warehouse program.	"BusinessViewVWPOutputTable.tag" on page 288

Table 129. Metadata templates supported by Visual Warehouse 5.2 (continued)

ConcurrentCascade.tag	Indicates that two business views are to be started at the same time.	"ConcurrentCascade.tag" on page 289
VWPProgramInstance	Modifies the definition of a Visual Warehouse program for use by a specific business view.	"VWPProgramInstance.tag" on page 291

BusinessView.tag

Use this template to define a business view. You must use this template if your partner application generates relationships between data sources and targets or contains programs that Visual Warehouse is to run.

The template also includes relationships to a security group, a subject area, and one or more agent sites.

Tokens

Table 130 provides information about each token in the template.

Table 130. BusinessView.tag tokens

Token	Description	Allowed values	Window or notebook: field
	Entity parameters		
**BVName*	Name of the business view. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**BVDescription*	Short description of the business view. This token is optional.	A text string, up to 200 bytes in length.	Business View: Description
**BVNotes*	Long description of the business view. This token is optional.	A text string, up to 32700 bytes in length.	Business View: Notes

Table 130. BusinessView.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**BVDataNotPresent*	Setting for how to handle warnings when the agent finds no data to extract for the business view. This token is required.	One of the following values: **ISV_BVDataNotPresent_OK** The business view is to successfully process if the agent finds no data to extract. **ISV_BVDataNotPresent_Warning** The business view is to fail if the agent finds no data to extract. **ISV_BVDataNotPresent_Error** The business view is to process with a warning if the agent finds no data to extract.	Business View: No Rows Returned Processing Options
**BVSelectStatementGenerated*	Flag indicating whether Visual Warehouse is to generate the SQL, or if the SQL is provided as the value of the **BVSelectStatement* token. This token is required.	One of the following values: **ISV_BVSELECTSTATEMENTYES** Visual Warehouse is to generate the SQL. **ISV_BVSELECTSTATEMENTNO** The SQL is provided as the value of the **BVSelectStatement* token.	None
**BVSelectStatement*	SQL statement to be executed. This token is required if **BVSelectStatementGenerated* is set to N.	An SQL statement, up to 32700 bytes in length.	Modify Business View SQL: SQL Statement
**BVContact*	Name of a person or group to contact for questions about this business view. This token is optional.	A text string, up to 64 bytes in length.	Business View: Admin Contact

Table 130. BusinessView.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**BVExternalPopulation*	Flag indicating whether an external application can populate the table. This token is required.	One of the following values: **ISV_BVEXTERNALYES** An external application can populate the table. **ISV_BVEXTERNALNO** Only Visual Warehouse can populate the table.	Business View: Externally Populated
**BVCreateTargetTable*	Flag indicating if Visual Warehouse is to create the target table when the business view is promoted to test status. This token is required.	One of the following values: **ISV_BVCREATETABLEYES** Visual Warehouse is to create the target table. **ISV_BVCREATETABLENO** Visual Warehouse is not to create the target table.	Business View: Visual Warehouse Created Table
**BVType*	Type of the business view. This token is required.	One of the following values: **ISV_BVType_EditionedAppend** Append a new edition of data to the target table each time the business view runs. **ISV_BVType_Full_Replace** Replace all the data in the target table each time the business view runs. **ISV_BVType_Uneditioned_Append** Append new data to the existing data each time the business view runs. **ISV_BVType_VWP_Population** Use a Visual Warehouse program to manage the data.	None

Table 130. BusinessView.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**BVSQLWarning*	Setting for whether the business view continues processing if an SQL warning code is issued. This token is required.	One of the following values: **ISV_BVSQLWarning_OK** The business view is to process successfully if an SQL warning code is issued. **ISV_BVSQLWarning_Warning** The business view is to process with a warning if an SQL warning code is issued. **ISV_BVSQLWarning_Error** The business view is to fail if an SQL warning code is issued.	Business View: SQL Warning Processing Options
Relationship parameters			
**SecurityGroup*	Security group in which to create all the objects being imported. This token is required, and you must specify the default security group.	`ISV_DEFAULTSECURITYGROUP` for the default security group.	Business View: Update Security Group
**SubjectArea*	Name of the group of business views. This token is required.	A text string, up to 80 bytes in length.	Subject: Name
**AgentSite*	Agent site to use for the business view: either a new agent site or the default agent site. This token is required, but you can specify the default agent site.	A text string, up to 80 bytes in length. Specify `ISV_DEFAULTAGENTSITE` for the default agent site.	Business View: Agent Site

Table 130. BusinessView.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**CurrentCheckPointID++*	Index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None

Examples of values

Table 131 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 131. Example values for BusinessView.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_7
**BVDescription*	This business view will extract Geography 7 data and write it to an UDB table
**BVNotes*	The Revenue for Geography 7 data comes from four source Oracle tables.
**BVDataNotPresent*	ISV_BVDataNotPresent_Warning1
**BVSelectStatementGenerated*	ISV_BVSELECTSTATEMENTNO
**BVSelectStatement*	"SELECT * FROM IWH.REVENUE_BY_GEOGRAPHY7"
**BVContact*	Greg Holland
**BVExternalPopulation*	ISV_BVEXTERNALNO
**BVCreateTargetTable*	ISV_CREATETABLEYES
**BVType*	ISV_BVType_VWP_Population
**BVSQLWarning*	ISV_BVSQLWarning_Error
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**Subject Area*	Group of business views generated for the partner tool
**AgentSite*	My agent site
**CurrentCheckPointID++*	10

BusinessViewInputTable.tag

Use this template to define a relationship between a business view and its source table. You can relate multiple source tables to the business view by reusing the template for each unique instance of a source table.

You must include this template for the following types of business views:

- Append editions (**BVType* is ISV_BVType_EditionedAppend)
- Replace existing data (**BVType* is ISV_BVType_Full_Replace)
- Append data without editions (**BVType* is ISV_BVType_Uneditioned_Append)

This template is optional for business views that use a Visual Warehouse program (**BVType* is ISV_BVType_VWP_Population).

Tokens

Table 132 provides information about each token in the template.

Table 132. BusinessViewInputTable.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	Name of the business view. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**DatabaseName*	Name of the database that contains the table. This token is required.	A text string, up to 80 bytes in length.	Information resource: Database
**TableOwner*	Owner, high-level qualifier, collection, or schema of the table. The owner must be a valid qualifier by the rules of ODBC. This token is required.	A text string, up to 15 bytes in length.	Table: Name Business View: Table Name Qualifier

Table 132. BusinessViewInputTable.tag tokens (continued). This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**TablePhysicalName*	Physical table name as defined to the database manager or file system. This token is required.	A text string, up to 80 bytes in length.	Table: Name Business View: Database Table Name
**CurrentCheckPointID++*	Index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None

Examples of values

Table 133 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 133. Example values for BusinessViewInputTable.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_1
**DatabaseName*	Operational_system_files
**TableOwner*	ISV_DEFAULTVALUE
**TablePhysicalName*	z:\geography\regions\geo1.file
**CurrentCheckPointID++*	13

BusinessViewOutputTable.tag

Use this template to define the relationship between a business view and its output target.

You must include this template for the following types of business views:

- Append editions (**BVType* is ISV_BVType_EditionedAppend)
- Replace existing data (**BVType* is ISV_BVType_Full_Replace)
- Append data without editions (**BVType* is ISV_BVType_Uneditioned_Append)

This template is optional for business views that use a Visual Warehouse program (**BVType* is ISV_BVType_VWP_Population).

Tokens

Table 134 provides information about each token in the template.

Table 134. BusinessViewOutputTable.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	Name of the business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**DatabaseName*	Name of the database that contains the table. This token is required.	A text string, up to 80 bytes in length.	Information resource: Database
**TableOwner*	Owner, high-level qualifier, collection, or schema of the table. This token is required.	A text string, up to 15 bytes in length.	Table: Name Business View: Table Name Qualifier
**TablePhysicalName*	Physical table name as defined to the database manager or file system. This token is required.	A text string, up to 80 bytes in length.	Table: Name Business View: Database Table Name
**CurrentCheckPointID++*	Index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None

Examples of values

Table 135 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 135. Example values for BusinessViewOutputTable.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_7
**DatabaseName*	Finance Warehouse
**TableOwner*	DB2ADMIN
**TablePhysicalName*	GEOGRAPHY

Table 135. Example values for BusinessViewOutputTable.tag tokens (continued)

Token	Example value
**CurrentCheckPointID++*	14

BusinessViewVWPOutputTable.tag

Use this template to define the relationship between a business view that uses a Visual Warehouse program and the output targets for the Visual Warehouse program.

Tokens

Table 136 provides information about each token in the template.

Table 136. BusinessViewVWPOutputTable.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	Name of the business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**DatabaseName*	Name of the database that contains the table. This token is required.	A text string, up to 80 bytes in length.	Information resource: Database
**TableOwner*	Owner, high-level qualifier, collection, or schema of the table. This token is required.	A text string, up to 15 bytes in length.	Table: Name Business View: Table Name Qualifier
**TablePhysicalName*	Physical table name as defined to the database manager or file system. This token is required.	A text string, up to 80 bytes in length.	Table: Name Business View: Database Table Name
**CurrentCheckPointID++*	Index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None

Examples of values

Table 137 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 137. Example values for VWPOutputTable.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_7
**DatabaseName*	Finance Warehouse
**TableOwner*	DB2ADMIN
**TablePhysicalName*	GEOGRAPHY
**CurrentCheckPointID++*	15

ConcurrentCascade.tag

Use this template to specify that Visual Warehouse is to start two business views at the same time. This template is required only if you want the business views to start at the same time.

Tokens

Table 138 provides information about each token in the template.

Table 138. ConcurrentCascade.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	Name of the business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**ConcurrentBVName*	Name of the business view that is to be started concurrently with the other business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Concurrently Starts: Business View Name
**CurrentCheckPointID++*	Index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None

Examples of values

Table 139 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 139. Example values for ConcurrentCascade.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_7
**ConcurrentBVName*	Revenue_by_Geography_6
**CurrentCheckPointID++*	16

PostCascade.tag

Use this template to identify that Visual Warehouse is to start another business view after the named business view finishes processing. This template is required only if you want to link business views in a cascaded relationship.

Tokens

Table 140 provides information about each token in the template.

Table 140. PostCascade.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	Name of the business view that is to finish processing before starting the next business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**PostBVName*	Name of the business view that is to start processing when the other business view finishes processing. This token is required.	A text string, up to 80 bytes in length.	Business View: Starts: Business View Name
**CurrentCheckPointID++*	Index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None

Examples of values

Table 141 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 141. Example values for PostCascade.tag tokens

Token	Example value
**BVName*	Revenue by geography 7
**PostBVName*	Revenue for all geographies
**CurrentCheckPointID++*	17

VWPProgramInstance.tag

Use this template to change the definition of a Visual Warehouse program for use by a specific business view. This template is required for each business view that uses the Visual Warehouse program.

Before using this template, you must define the base definition of the Visual Warehouse program in VWPProgramTemplate.tag (see page 93). This template defines the relationship to the base Visual Warehouse program definition (VWPProgramTemplate.tag) as well as to the business view that uses the Visual Warehouse program.

Tokens

Table 142 provides information about each token in the template.

Table 142. VWPProgramInstance.tag tokens

Token	Description	Allowed values	Window or notebook: field
		Entity parameters	
**VWPInstanceNotes*	Long description of the Visual Warehouse program and what it does. This token is optional.	A text string, up to 32700 bytes in length.	None

Table 142. VWPProgramInstance.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**VWPProgramInstanceKey*	Key that uniquely identifies this program instance. The key must be unique from all other keys in the tag language file. **Tip:** Finish processing the VWPProgramInstance.tag template before increasing the value of the key. This token is required.	A numeric value.	None
	Relationship parameters		
**BVName*	Name of the business view. This token is required.		Business View: Business Name
**VWPProgramTemplateName*	Name of the parent Visual Warehouse program template for this Visual Warehouse program instance. This token is required.	A text string, up to 80 bytes in length.	Program: Business Name
**CurrentCheckPointID++*	Index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None

Examples of values

Table 143 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 143. Example values for VWPProgramInstance.tag tokens

Token	Example value
**VWPInstanceNotes*	This program exports data from the Geography database.

Table 143. Example values for VWPProgramInstance.tag tokens (continued)

Token	**Example value**
**VWPProgramInstanceKey*	070000
**BVName*	Revenue by geography
**VWPProgramTemplateName*	My partner program
**CurrentCheckPointID++*	11

Appendix C. Writing your own program to use with the Data Warehouse Center

You can write Data Warehouse Center programs in any language that supports one of the following program types: executable, batch program, or dynamic link library.

If the program has a program type of executable, command file, or dynamic link library, it must reside on the agent site. The Data Warehouse Center agent starts the program at the scheduled time. On Windows NT and Windows 2000, the agent runs as a system process by default. The program cannot access resources or programs that require a user ID. Also, any environment variables that the program needs to access must be system variables.

To change the Data Warehouse Center server, logger, and agent daemon processes to run as user processes:

1. Double-click the **Services** icon in the **Control Panel** folder.
2. Stop the Agent service.
3. Select the Agent service and click **Startup**.
4. Click **This Account**.
5. Click the push button after the **This Account** field to select a user ID.

   The user ID must have administrator authority in Windows NT or Windows 2000 and authorization to any required network drive.
6. Type the password for the user ID twice.
7. Click **OK**.
8. Restart the workstation.

If you write programs that use Object REXX, complete the following procedure to enable these programs to run on Windows NT or Windows 2000:

1. Define the Data Warehouse Center agent or server service as a system process that can interact with the Windows NT or Windows 2000 desktop:
   a. Select the agent or server service from the **Service** list.
   b. Click **Startup**.
   c. Click **System Account**.
   d. Select the **Allow Service to Interact with Desktop** check box.
2. Initialize the Object REXX environment before the agent or server starts the program. You can initialize the environment by running any Object REXX program from the command line.

3. If your Object REXX program issues a DB2 CONNECT statement, verify that the statement includes the user ID and password, as in the following example:

   ```
   DB2 CONNECT TO testdb USER vwadmin USING vwpass
   ```

Passing parameters

At run time, the Data Warehouse Center generates a command-line parameter list that it passes as input to your program. Whenever possible, test your program from the command line before using it in a step.

Example: The Data Warehouse Center program VW 5.2 DB2 load replace (VWPLOADR) selects data from a file and loads the data into a database. It uses the following parameters:

- Source file name
- Target database name
- Target database user ID
- Target database password
- Target table name
- Column delimiter

The program gets the parameters as shown in Figure 75:

```
char * sourceFile;
     sourceFile = argv[1]:
     char * dbName;
     dbName = argv[2];
     char * dbUser;
     dbUser = argv[3];
     char * dbPassword
     dbPassword = argv[4];
     char * dbTable;
     dbTable = argv[5]
     char * fileMod;
     if(argc>6) fileMod = argv[6];
     else fileMod = NULL;
```

Figure 75. Reading parameters from the command line

The program uses the target parameters to connect to the target database, as shown in Figure 76 on page 297:

```
rc = SQLConnect (hdbc, (SQLCHAR *)dbName, SQL_NTS,
           (SQLCHAR *)dbUser, SQL_NTS,       /* UID */
           (SQLCHAR *)dbPassword, SQL_NTS);  /* Password */
```

Figure 76. Connecting to the target database

The program then uses the DB2 load utility to load data into the database.

Returning status information

After your Data Warehouse Center program runs, it must return a return code to the step that uses the program. The return code must be a positive integer. If your program does not return a return code, the step using the program fails. The Data Warehouse Center displays the return code in the **Error RC2** field of the Log Details window when the value of **Error RC1** is 8410.

Your Data Warehouse Center program can return additional status information to the Data Warehouse Center:

- Another return code, which can be the same as or different from the code that is returned by the Data Warehouse Center program.
- A warning flag that indicates that the Data Warehouse Center is to treat the return code as a warning. When your program sets this flag, the step that uses this program will have Warning status in the Operations Work in Progress window.
- A message, which is displayed in the **System Message** field of the Log Viewer Details window
- The number of rows of data that the program processed.

  The Data Warehouse Center displays the number in the Log Viewer Details window for the step.
- The number of bytes of data that the program processed.

  The Data Warehouse Center displays the number in the Log Viewer Details window for the step.
- The SQLSTATE return code, which the Data Warehouse Center displays in the SQL state field of the Log Viewer Details window.

The Data Warehouse Center agent transfers the additional status information to the warehouse server.

Transferring the information to the Data Warehouse Center

To transfer the additional status information to the warehouse agent, your program must create a file, called a *feedback file*, containing the additional status information. The path and file name for the feedback file must be the value of the VWP_LOG environment variable. (The file name is *processid*.log, where *processid* is the ID of the agent process.) The agent sets VWP_LOG before it calls the program. After the program finishes running, the agent checks whether the feedback file exists. If it exists, the agent processes the file. Otherwise, the agent will do nothing. If the program cannot create the file, it should continue to run.

Format of the feedback file

Your program can write the additional status information to the feedback file in any order, but must use the following format to identify information. Enclose each returned item within the begin tag <tag> and end tag </tag> in the following list. Each begin tag must be followed by its end tag; you cannot include two begin tags in a row. For example, the following tag format is valid:

<RC>...</RC>...<MSG>...</MSG>

The following embedded tag format is not valid:

<RC>...<MSG>...</RC>...</MSG>

You can specify the following information in the feedback file:

Return code
: <RC>*return code*</RC>, where *return code* is a positive integer.

Return code warning flag
: <WARNING>1</WARNING> sets the return code warning flag to On.

Data Warehouse Center system message
: <MSG>*message text*\n</MSG>

  message text
  : The text of one or more messages

  \n
  : The new line character. Include this character at the end of each message if there are multiple messages.

Comment
: <COMMENT>*comment text*</COMMENT>, where *comment text* is the text of the comment.

Number of rows of data processed
: <ROWS>*number of rows*</ROWS>, where *number of rows* is any positive integer.

Number of bytes processed
: <BYTES>*number of bytes*</BYTES>, where *number of bytes* is any positive integer.

SQLSTATE
<SQLSTATE>*sqlstate string*</SQLSTATE>, where *sqlstate string* is any string whose length is greater than 0 and less than or equal to 5 digits.

Figure 77 shows an example of the feedback file.

```
<RC> 20</RC>
<ROWS>2345</ROWS>
<MSG>The parameter type is not correct</MSG>
<COMMENT> Please supply the correct parameter type (PASSWORD
     NOTREQUIRED, GETPASSWORD, ENTERPASSWORD)</COMMENT>
<BYTES> 123456</BYTES>
<WARNING> 1</WARNING>
<SQLSTATE>12345</SQLSTATE>
```

Figure 77. Example of the feedback file

How the feedback determines the step status

The return codes and step status for the program that are displayed in the Log Viewer vary. They depend on the following values set by the program:

- The value of the return code that the program returned
- Whether a feedback file exists
- The value of the return code in the feedback file
- Whether the warning flag is set to On

Table 144 on page 300 lists the possible combinations of these values and the results that they produce.

Table 144. Feedback file conditions and results

Conditions				**Results**	
				Step status[1]	**Values of Error RC1 and RC2**
Data Warehouse Center program return code is 0	No feedback file exists[2]			Successful	**RC1** = 0; **RC2** = 0
	A feedback file exists[2]	The value of <RC> in the feedback file is 0[3]	<WARNING> is not set in the feedback file	Successful	**RC1** = 0; **RC2** = 0
			The value of <WARNING> in the feedback file is 1	Warning	**RC1** = 0; **RC2** = 0
		The value of <RC> in the feedback file is non-0[3]	<WARNING> is not set in the feedback file	Failed	**RC1** = 8410 (the program failed); **RC2** = the value of <RC> in the feedback file
			The value of <WARNING> in the feedback file is 1	Warning	**RC1** = 0; **RC2** = the value of <RC> in the feedback file

Table 144. Feedback file conditions and results (continued)

Conditions				**Results**	
				Step status[1]	**Values of Error RC1 and RC2**
The Data Warehouse Center program return code is nonzero	No feedback file exists[2]			Failed	**RC1** = 8410 (the Data Warehouse Center program failed); **RC2** = the code returned by the Data Warehouse Center program
	A feedback file exists[2]	The value of <RC> in the feedback file is 0[3]	<WARNING> is not set in the feedback file	Successful	**RC1** = 0; **RC2** = 0
			The value of <WARNING> in the feedback file is 1	Warning	**RC1** = 0; **RC2** = 0
		The value of <RC> in the feedback file is non-0	<WARNING> is not set in the feedback file	Failed	**RC1** = 8410 (the Data Warehouse Center program failed); **RC2** = the code returned by the Data Warehouse Center program
			The value of <WARNING> in the feedback file is 1	Warning	**RC1** = 0; **RC2** = the value of <RC> in the feedback file

Table 144. Feedback file conditions and results (continued)

Conditions	**Results**	
	Step status[1]	**Values of Error RC1 and RC2**

Notes:

1. The step processing status, which is displayed in the Work in Progress window.
2. The Data Warehouse Center checks for the existence of the feedback file, regardless of whether the return code for the program is 0 or nonzero.
3. The Data Warehouse Center always displays the value of <RC> in the feedback file as the value of the **RC2** field in the Log Details window.

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

Trademarks

The following terms, which may be denoted by an asterisk(*), are trademarks of International Business Machines Corporation in the United States, other countries, or both.

ACF/VTAM	IBM
AISPO	IMS
AIX	IMS/ESA
AIX/6000	LAN DistanceMVS
AIXwindows	MVS/ESA
AnyNet	MVS/XA
APPN	Net.Data
AS/400	OS/2
BookManager	OS/390
CICS	OS/400
C Set++	PowerPC
C/370	QBIC
DATABASE 2	QMF
DataHub	RACF
DataJoiner	RISC System/6000
DataPropagator	RS/6000
DataRefresher	S/370
DB2	SP
DB2 Connect	SQL/DS
DB2 Extenders	SQL/400
DB2 OLAP Server	System/370
DB2 Universal Database	System/390
Distributed Relational	SystemView
Database Architecture	VisualAge
DRDA	VM/ESA
eNetwork	VSE/ESA
Extended Services	VTAM
FFST	WebExplorer
First Failure Support Technology	WIN-OS/2

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, other countries, or both.

Bibliography

For information about how to use the Data Warehouse Center, see the online help. The Data Warehouse Center provides help for specific windows and for general tasks, such as creating warehouse sources and steps.

For information about IBM products that are related to the Data Warehouse Center, go to the IBM Data Management Web site at http://www.software.ibm.com/data/

The Data Warehouse Center library includes the following publications:

IBM DB2: DB2 Warehouse Manager Installation Guide, SC26-3496
IBM DB2: Messages and Reason Codes (HTML book that is included in the Data Warehouse Center folder)
IBM DB2: Information Catalog Manager Administration Guide, SC26-3362
IBM DB2: Information Catalog Manager Programming Guide and Reference, SC26-3368
IBM DB2 OLAP Server: Using DB2 OLAP Server, SC26-9235

Index

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. This guide suggests information that you can gather to help DB2 Customer Support to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

If you live in the U.S.A., then you can call one of the following numbers:
- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

ftp.software.ibm.com

Log on as anonymous. In the directory `/ps/products/db2`, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l

These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2

Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the *IBM Software Support Handbook*. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Part Number: CT60KNA

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9994-00

CT60KNA

Spine information:

IBM® DB2® Universal Database

Data Warehouse Center Application Integration Guide

Version 7