IBM Visual Warehouse for Windows NT

Integrating Applications with the Visual Warehouse Solution

Version 5 Release 2

IBM Visual Warehouse for Windows NT

Integrating Applications with the Visual Warehouse Solution

Version 5 Release 2

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 297.

First Edition (November 1998)

This edition applies to Version 5 Release 2 of Visual Warehouse, 5639–VW5, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this book

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

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 Visual Warehouse, DataGuide, or both.

Prerequisite knowledge

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 Visual Warehouse and DataGuide before you use the integration features described in this document. Specifically, you must know how to do the tasks listed in the following table:

Task	**For more information, see:**
Create an information catalog in DataGuide	*Managing DataGuide*
Import and export metadata	*Managing DataGuide*
Define a Visual Warehouse agent site	*Managing Visual Warehouse* and the Visual Warehouse online help
Create, promote, run, and monitor business views	*Managing Visual Warehouse* and the Visual Warehouse online help
Create Visual Warehouse programs and use them in a business view	*Managing Visual Warehouse* and the Visual Warehouse online help
Modify parameters for Visual Warehouse programs	*Managing Visual Warehouse* and the Visual Warehouse online help
Import and export metadata	*Managing Visual Warehouse* and the Visual Warehouse online help

For a list of publications for Visual Warehouse and DataGuide, see "Bibliography" on page 299.

How to send your comments

Your feedback is important in helping to provide accurate and high-quality information. If you have any comments about this book or any other Visual Warehouse documentation, visit the following Web site:

```
http://www.software.ibm.com/data/vw
```

There you will find a feedback page where you can enter and submit your comments.

Part 1. Integrating applications with the Visual Warehouse solution

Chapter 1. Planning to integrate your applications

You can use the Visual Warehouse solution to bring together various applications that help users build and manage a data warehouse. You can use Visual Warehouse to identify the data that you want to manage. You can use Visual Warehouse to transform that data into information that will be meaningful for data warehouse users.

You can use Visual Warehouse 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 data warehousing application
- Scheduling extracts and transformations of data that is based on the date and time or on an event
- Publishing metadata for data warehouse users to use

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

How partner applications can work with Visual Warehouse and DataGuide

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

For example, you want to unload operational data from an IMS™ database, clean the data, and load the cleansed data into a DB2® warehouse database. End 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 data 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 end users can search for specific attributes. End users use the metadata to determine which tables have the data 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.

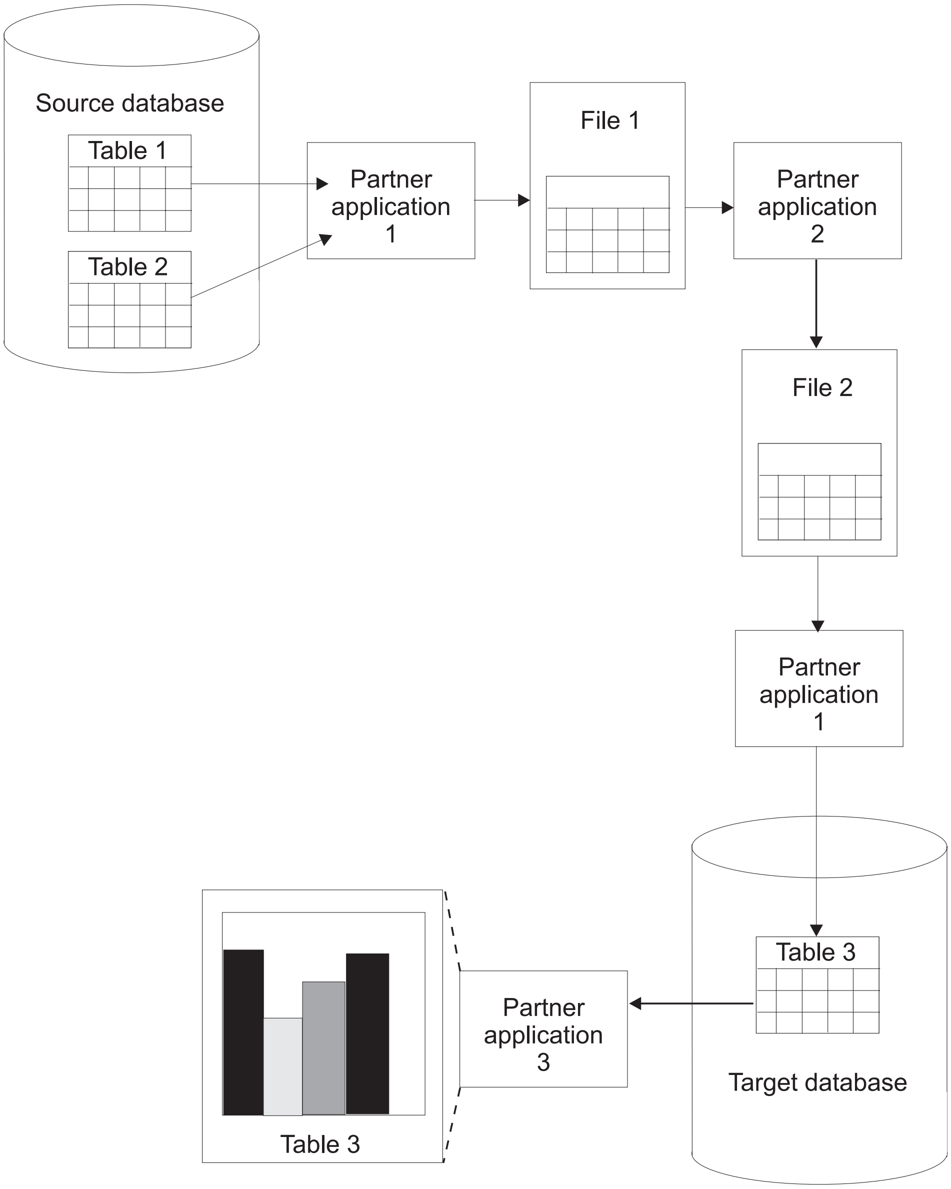


Figure 1. Using partner applications together to build a data warehouse

Managing partner applications

You can use Visual Warehouse business views to manage this process. A *business view* is a step in the transformation process from the data's source format to its target format. You use business views to define and schedule each step in the extraction, transformation, and writing of the data.

A basic business view performs the following tasks:

- It extracts data from at least one table or file.
- It uses Visual Warehouse's 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 business views, one for each source-to-target transformation:

- The Unload business view performs steps 1 through 3.
- The Clean business view performs steps 4 through 6.
- The Load business view performs steps 7 through 8.

Because Partner application 3 does not transform data in step 9, you do not define a business view for step 9.

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

You can schedule the first business view to run at a particular date and time. You can schedule the second business view to start after the first business view runs. You schedule the third business view to start after the second business view 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 Visual Warehouse. In this book, *partner metadata* is metadata that partner applications use and store outside of Visual Warehouse.

In the partner application example, you import the following metadata into Visual Warehouse:

- From Partner application 1, metadata about the databases, File 1, and the application itself

- From Partner application 2, metadata about File 2 and the application itself

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 Visual Warehouse to DataGuide to provide information about the data in the warehouse to end users of the data warehouse. You can import metadata for the data sources and targets, as well as the transformations of the data from its source format to its target format. The end users of your data warehouse can obtain information about the lineage of the data in the data warehouse from the metadata that you import.

In the partner application example, you export metadata about the table in the data warehouse, Table 3, to DataGuide®.

You can import metadata into DataGuide directly from Visual Warehouse. You can also import metadata into DataGuide if the partner applications support metadata in MDIS format.

You can export metadata from DataGuide to a partner metadata store. In the partner application example, you export metadata about Table 3 from DataGuide to the metadata store for Partner application 3. End 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, Visual Warehouse, and DataGuide.

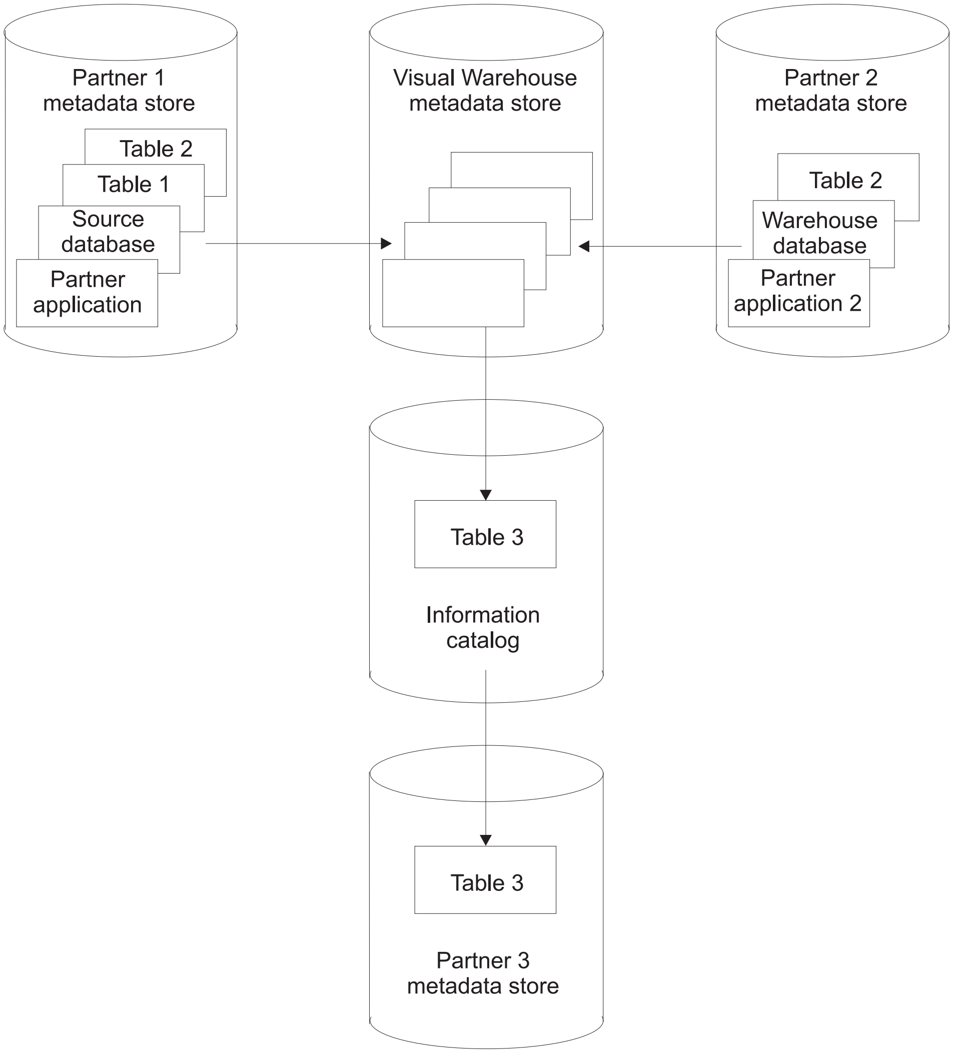


Figure 2. The flow of metadata among the partner applications, Visual Warehouse, and DataGuide

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

- For more information about importing metadata into Visual Warehouse, see "Chapter 2. Importing metadata into Visual Warehouse" on page 11.
- For more information about exporting metadata from Visual Warehouse, see "Chapter 3. Exporting metadata from Visual Warehouse" on page 35.
- For more information about importing metadata into DataGuide, see "Chapter 5. Importing metadata into DataGuide" on page 47.

- For more information about exporting metadata from DataGuide, see "Chapter 4. Exporting metadata from DataGuide" on page 39.

Integration scenarios

Table 1 lists some common types of data warehousing applications and describes how you can integrate them with Visual Warehouse.

Table 1. Integration scenarios

Type of application	Integration process
Data warehousing design	To use data from data warehousing design applications in Visual Warehouse: 1. Import run-time metadata into Visual Warehouse. 2. Use metadata synchronization to propagate metadata into DataGuide.
Operational data descriptions	Import run-time metadata into Visual Warehouse and business metadata into DataGuide. 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 DataGuide directly.
Data cleansing	To clean operational data: 1. Determine which application will manage the movement of the source data and target data: Visual Warehouse or the partner application. Different applications can manage the source data and the target data. 2. Import data source and data target definitions, or export data source and data target definitions, or both. Do so to avoid typing the definitions again. 3. Define the partner application as a Visual Warehouse program, or write a Visual Warehouse program that starts the partner application. 4. Develop a user interface that sets the partner application parameters. 5. Import the run-time metadata into Visual Warehouse so that Visual Warehouse can run the data cleansing application. You can schedule programs by sequence as well as date and time. 6. Import business metadata into DataGuide for use by end users.

Table 1. Integration scenarios (continued)

Type of application	Integration process
Alternate data storage (such as DB2 OLAP Server™)	To load operational data into alternate data storage: 1. From Visual Warehouse. export the data definitions needed to build the partner storage. 2. Define the load programs as a Visual Warehouse program or write a Visual Warehouse 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 Visual Warehouse. 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 DataGuide.
Reporting	To integrate reporting applications with Visual Warehouse: 1. Export business metadata from DataGuide into the report application. 2. Enable launch of the report application from an information catalog. 3. Import descriptions of the reports into DataGuide.

Hardware requirements and software requirements

The models and templates that are described in this book require Visual Warehouse Version 5.2, DataGuide Administrator Version 5.2, and their prerequisite products.

For information about the prerequisite products for Visual Warehouse and DataGuide, see *Installing Visual Warehouse and DataGuide*.

Chapter 2. Importing metadata into Visual Warehouse

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

To import metadata into Visual Warehouse:

1. Build a *tag language file* (a file that contains the metadata for the objects to import).
2. Import the tag language file.

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 Visual Warehouse metadata templates. *Visual Warehouse 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 having to refer 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 Visual Warehouse:

Agent sites

A *Visual Warehouse agent* performs the actual transfer of data between the source database or file, and the target database. It also performs any transformation of that data. The Visual Warehouse agent receives commands from the Visual Warehouse server. Then, the agent issues SQL commands, starts a partner application, or starts a Visual Warehouse program that starts a partner application. A Visual 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 either the source database or the target database.

Source databases and target databases and files

A *source database* or *source file* is the database or file from which Visual Warehouse or a partner application extracts data for further processing. The generic term *data source* means a source database, source file, or both. A source database is associated with one or more tables. A table or file is associated with one or more columns or fields.

A *target database* or *target file* is the database or file to which Visual Warehouse or a partner application writes the data after processing it. The generic term *data target* means a target database, target file, or both. A target database is associated with one or more tables. A table or file is associated with one or more columns or fields.

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

Visual Warehouse programs

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

Business views

A business view is a step in the transformation process from the data's source format to its target format. The metadata for the business view includes the source and target tables on which Visual Warehouse or the partner application is to operate. It also includes the SQL to issue or the program to start to perform the transformation.

Subjects

A *subject* is a logical collection of business views that does not need to correspond to a physical database. For example, you have a series of business views that work together, using Visual Warehouse programs, to unload, clean, and load data. You can use a subject to identify these programs as a set of programs.

Cascade relationships between business views

A *cascade relationship* is a schedule for a business view that is based on the processing status of another business view. You can schedule a business view to run after another business view finishes running or schedule two business views to run simultaneously.

Relationships between Visual Warehouse objects

The metadata for Visual Warehouse objects describes relationships to other objects. For example, the metadata for a business view describes relationships to the source and target tables that the business view uses.

Defining objects with Visual Warehouse metadata templates

To define objects that you want to import into Visual Warehouse, you build a language file from one or more Visual Warehouse 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 install the templates when you install the entire product, the Visual Warehouse server, or the Visual Warehouse agent. You can also install the templates alone.

For information about installing the entire product, the Visual Warehouse server, or the Visual Warehouse agent, see *Planning and Installing Visual Warehouse and DataGuide.*

To install the templates alone:

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

   The default directory for the ISV Toolkit is x:\vwswin\templates. Visual Warehouse 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.

Visual Warehouse installs the files in subdirectories of the directory that is set by *VWS_TEMPLATES*. Table 2 on page 14 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 Visual Warehouse 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 have to recompile your program.

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

1. Use the *ISV_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 the value with which you want to replace the*DatabaseNotes token is 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 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.

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. Visual Warehouse will write these checkpoints to the log file as the tag language file is being imported.

Appending templates to the tag language file: Figure 3 on page 16 shows the order of the templates in the tag language file and the relationship between templates.

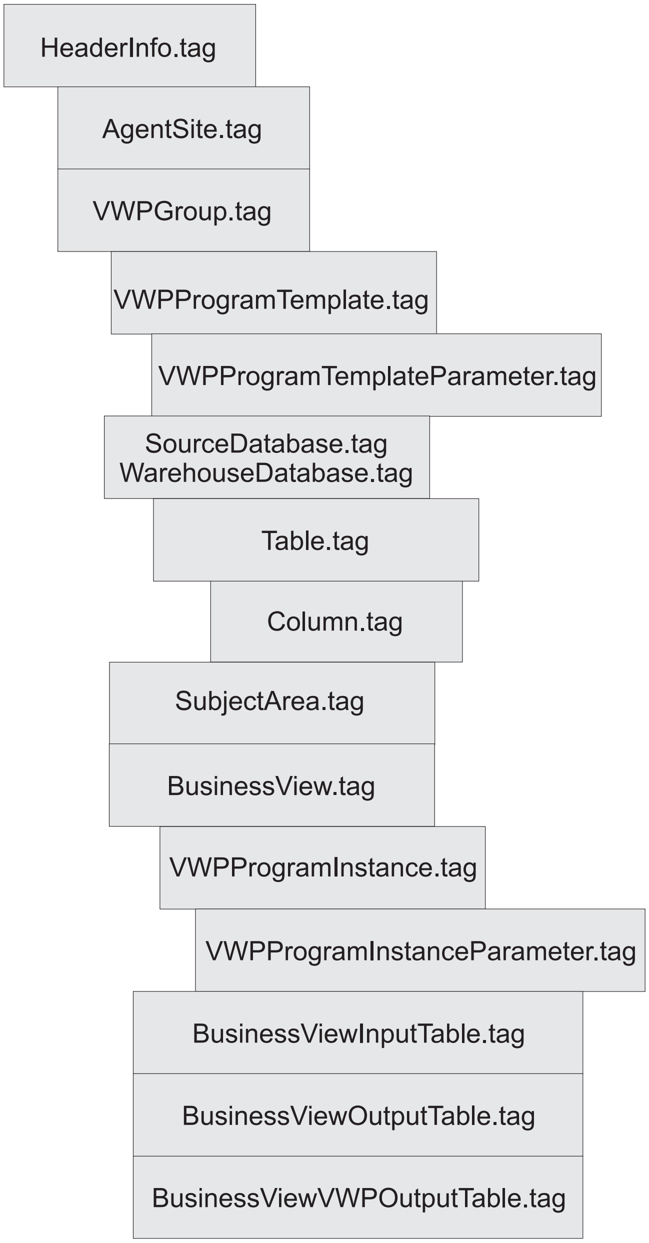


Figure 3. Relationship between templates in the tag language file. Indented templates have a relationship with the template above it.

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 lists the order in which your program must append templates to the tag language file. It also provides the conditions under which the template is required or optional.

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 Visual Warehouse programs
4	VWPProgramTemplate.tag	Required if you are defining Visual Warehouse programs
5	VWPProgramTemplateParameter.tag	Required if you are defining Visual Warehouse programs
6	SourceDataBase.tag WarehouseDataBase.tag	Required if you are defining data sources or targets
7	Table.tag	Required if you are defining data sources or targets
8	Column.tag	Required if you are defining data sources or targets
9	SubjectArea.tag	Required if you are defining business views
10	BusinessView.tag	Required if you are defining business views
11	VWPProgramInstance.tag	Required if the business view uses a Visual Warehouse program
12	VWPProgramInstanceParameter.tag	Required if the business view uses a Visual Warehouse program
13	BusinessViewInputTable.tag	Required if the business view uses SQL to write data to a target table in the warehouse database.

Table 3. Relationships between templates and conditions (continued)

Order	Template	Required or optional?
14	BusinessViewOutputTable.tag	Required if the business view uses SQL to write data to a target table in the warehouse database.
15	BusinessViewVWPOutputTable.tag	Optional if the business view uses a Visual Warehouse program.
16	ConcurrentCascade.tag PostCascade.tag	Required to start a business view based on the processing status of another business view

For detailed information about these templates, see "Chapter 7. Metadata templates" on page 63.

Logging processing messages: Your interchange program can write log processing messages or trace files to the directory that the *VWS_LOGGING* environment variable specifies. Visual Warehouse 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

Your program must substitute the values it supplies for the tokens in the template. For information about the tokens in the template, see "HeaderInfo.tag" on page 84.

Program logic: Figure 4 is a pseudocode example of the logic your program can use to build the header portion of the tag language file.

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

Figure 4. Pseudocode of adding the header to the tag language file

The ISV_Sample program provides an example of building 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 Visual 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 Visual Warehouse agent is installed. Your program must substitute the values it obtains for the appropriate tokens in the template.

Program logic

Figure 5 shows a pseudocode example of the logic your program can use to add a new agent site to the tag language file.

```
If you want to create an AgentSite specific to the partner application:
     Read a copy of the AgentSite.tag template from the ISV directory
     Search for and replace tokens with partner metadata (or defaults)
     Append the output to a target file
Else
     Set AgentSite token to default agent site value
```

Figure 5. 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 data sources and data targets

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

- The data source or data target is already in the Visual Warehouse control database.
- You are using only the business views that use Visual Warehouse programs.

To define data sources and data targets:

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

Copying templates

You can define the following types of data source objects:

- Relational databases

- IMS databases
- File systems
- Files

You can define relational databases as data target objects.

Tables 4 through 5 list the templates that your program must copy and change to define each type of data source and data target object.

Relational tables: Table 4 lists the templates that your program must copy to define a relational database.

Table 4. 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	SourceDataBase.tag (see page 86) WarehouseDataBase.tag (see page 107)	HeaderInfo.tag (see page 84) AgentSite.tag (see page 65) if you are not using the default agent
Table	One copy for each table that you want to use in the database	Table.tag (see page 91)	SourceDataBase.tag (see page 86) WarehouseDataBase.tag (see page 107)
Column	One copy for each column that you want to use in each table	Column.tag (see page 77)	Table.tag (see page 91)

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 6 shows the relationship between the database, table, and column templates.

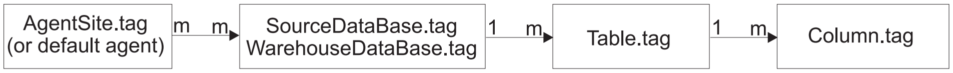


Figure 6. Relationship between the DataBase.tag, Table.tag, and Column.tag templates

IMS databases: Table 5 on page 22 lists the templates that your program must copy to define an IMS database.

Table 5. 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	SourceDataBase.tag (see page 86)	HeaderInfo.tag (see page 84) AgentSite.tag (see page 65) if you are not using the default agent
Segment	One copy for each segment that you want to use in the database	Table.tag (see page 91)	SourceDataBase.tag (see page 86)
Field	One copy for each field that you want to use in each segment	Column.tag (see page 77)	Table.tag (see page 91)

You define relationships between the templates for the database, segments, and fields in the same manner that you define relationships for tables. (See Figure 6 on page 21.)

Files: Table 5 lists the templates that your program must copy to define either a file system and its associated files, or a single file.

Table 6. Templates for file systems or a single file

Source or target definition	Number of copies of template	Template to copy	Prerequisite template
File system or single file	One copy for each file system or single file	SourceDataBase.tag (see page 86)	HeaderInfo.tag (see page 84) AgentSite.tag (see page 65) if you are not using the default agent
File	One copy for each file that you want to use in the file system or for each single file	Table.tag (see page 91)	SourceDataBase.tag (see page 86)
Field	One copy for each field that you want to use in each file	Column.tag (see page 77)	Table.tag (see page 91)

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 6 on page 21.)

Substituting values

Your program must obtain values that describe databases or files from the partner metadata store. Your program must substitute the values 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 that 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 7 on page 24 shows a pseudocode example of the logic that your program can use to define source databases and files, and target databases.

```
For each source database to define:
     Read a copy of the SourceDatabase.tag template
     Search for and replace tokens with partner metadata (or defaults)
     Append the output to a target file

     For each table, file, or segment to define:
          Read a copy of the Table.tag template
          Search for and replace tokens with partner metadata (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 partner metadata (or defaults)
               Append the output to a target file
          End (for each column)
     End (for each table)
End (for each source database)

For each warehouse database to define:
     Read a copy of the WarehouseDatabase.tag template
     Search for and replace tokens with partner metadata (or defaults)
     Append the output to a target file

     For each table, file, or segment to define:
          Read a copy of the Table.tag template
          Search for and replace tokens with partner metadata (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 partner metadata (or defaults)
               Append the output to a target file
          End (for each column)
     End (for each table)
End (for each warehouse database)
```

Figure 7. Pseudocode for defining source and target databases and their associated tables and columns. Use this logic for each source or target database that you want to define.

The ISV_Sample program provides an example of defining source and target databases and files, and their associated tables and columns. 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 Visual Warehouse programs

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

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

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

   You can change the parameters that are used in a particular business view by defining an instance of the program parameters for the business view. For more information about using a Visual Warehouse program in a business view, see “Defining business views” on page 27.

For information about writing a program for use with Visual Warehouse, see “Appendix B. Writing your own program for use with Visual Warehouse” on page 289.

To define a Visual Warehouse program:

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

Copying templates

Table 7 lists the templates your program must copy and change to define Visual Warehouse programs.

Table 7. Templates for Visual Warehouse programs

Definition	Number of copies of template	Template to copy	Prerequisite template
Visual Warehouse program group	One copy for each group to define	VWPGroup.tag (see page 95)	HeaderInfo.tag (see page 84)
Visual Warehouse program template	One copy for each Visual Warehouse program to define	VWPProgramTemplate.tag (see page 101)	VWPGroup.tag (see page 95)
Visual Warehouse program template parameter	One copy for each parameter in each Visual Warehouse program	VWPProgramTemplateParameter.tag (see page 104)	VWPProgramTemplate.tag (see page 101)

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

Figure 8 shows the relationship between the Visual Warehouse program group, Visual Warehouse program, and the Visual Warehouse program parameters.

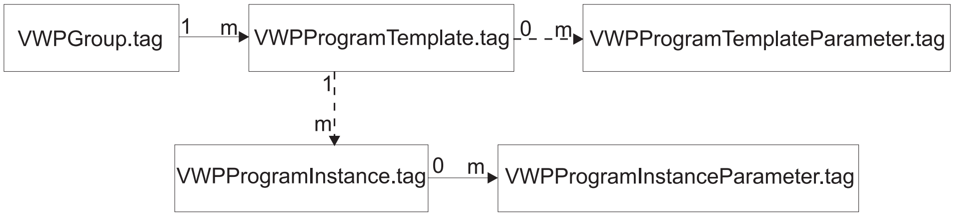


Figure 8. Relationship between the VWPGroup.tag, VWPProgramTemplate.tag, and VWPProgramTemplateParameter.tag templates

For information about relating a Visual Warehouse program to a business view, see “Defining business views” on page 27.

Substituting values

Your program must obtain values that describe the Visual Warehouse programs from the partner metadata store:

- The Visual Warehouse program groups to define
- The Visual Warehouse programs to define
- The parameters in each Visual Warehouse program to define

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

Program logic

Figure 9 on page 27 shows a pseudocode example of the logic that your program can use to define the partner application.

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

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

     For each parameter to pass to the partner application:
          Read a copy of the VWPProgramTemplate.tag template
          Search for and replace tokens with partner metadata (or defaults)
          Append the output to a target file
     End (for each parameter)
End (for each application)
```

Figure 9. Pseudocode for defining Visual Warehouse programs

The ISV_Sample program provides an example of adding Visual Warehouse 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 business views

A business view is a step in the transformation process from the data's source format to its target format. You must define a business view for each step in the transformation process that you want Visual Warehouse to manage. Use the information in this section to determine how to define your business views, rather than the information in the Visual Warehouse online help. The templates require different relationships from business views that are defined using the user interface.

You must define a subject for the business views. You can use subjects to group business views that use a particular partner application.

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

1. One or more subjects to contain the business views.
2. One or more business views.
3. For each business view, a relationship to one or more source tables and a target table if the business view uses SQL to do the source-target mapping. If the business view uses a Visual Warehouse program, the source tables and target table are optional.
4. If the business view uses a Visual Warehouse program:
   a. An instance of the Visual Warehouse program

b. The parameters associated with the Visual Warehouse program
c. Optionally, the output table for the Visual Warehouse program

To define business views:

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

Copying templates

Table 8 lists the templates that your program must copy and change to define business views.

Table 8. Templates for business views

Definition	Number of copies of template	Template to copy	Prerequisite template
Subject	One copy for each subject	SubjectArea.tag (see page 89)	HeaderInfo.tag (see page 84) AgentSite.tag (see page 65) if you are not using the default agent
Business view	One copy for each business view	BusinessView.tag (see page 67)	SubjectArea.tag (see page 89)
Source table for the business view	One copy for each source table for the business view	BusinessViewInputTable.tag (see page 72)	Table.tag (see page 91)
Target table for the business view	One copy if the business view has a target table	BusinessViewVWPOutputTable.tag (see page 75)	Table.tag (see page 91)
Visual Warehouse program instance	One copy if the business view uses a Visual Warehouse program	VWPProgramInstance.tag (see page 97)	VWPProgramTemplate.tag (see page 101)
Visual Warehouse program instance parameters	One copy for each parameter used in the business view	VWPProgramInstanceParameter.tag (see page 99)	VWPProgramTemplateParameter.tag (see page 104)

You relate the templates for the subject to the templates for the business views by specifying common values in the templates. Similarly, you relate templates for the business views to the templates for input tables and output tables by specifying common values in the templates. You can also relate the template for the business view to a template for the Visual Warehouse program by specifying common values in the templates.

Figure 10 shows the relationship between the subject, business view, input tables, output table, Visual Warehouse program instance, and the Visual Warehouse program instance parameters.

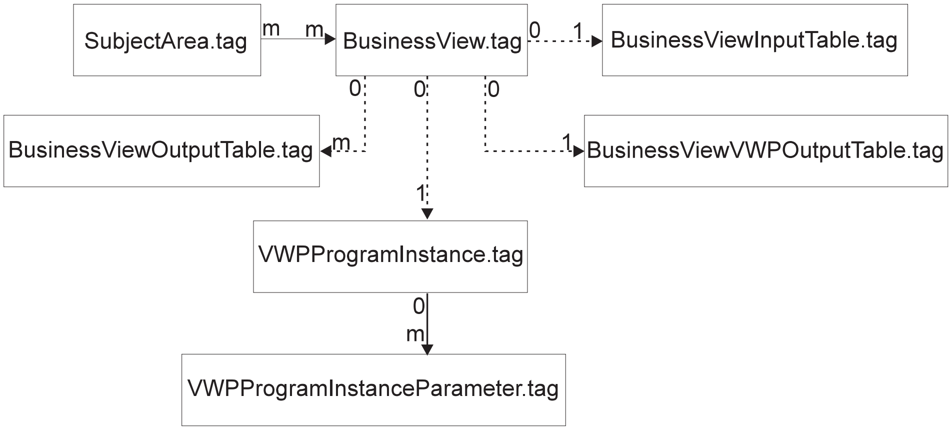

SubjectArea.tag
m
m
BusinessView.tag
0
1
BusinessViewInputTable.tag
0
0
0
BusinessViewOutputTable.tag
m
1
BusinessViewVWPOutputTable.tag
1
VWPProgramInstance.tag
0
m
VWPProgramInstanceParameter.tag

Figure 10. Relationship between the SubjectArea.tag, BusinessView.tag, BusinessViewInputTable.tag, BusinessViewOutputTable.tag, BusinessViewVWPOutputTable.tag,VWPProgramInstance.tag, and VWPProgramInstanceParameter.tag templates. See Figure 8 on page 26 to see how the Visual Warehouse program instance templates relate to the other Visual Warehouse program templates.

Substituting values

Your program must obtain values that describe the subjects and business views from the partner metadata store:

- The subjects to contain the business views
- The business views to define
- The source tables for each business view
- The target table for each business view
- The Visual Warehouse program and parameters for the business view, if applicable

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 to build the portion of the tag language file for the business views.

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

For each business view to define:
     Read a copy of the BusinessView.tag template
     Search for and replace tokens with partner metadata (or defaults)
     Append the output to a target file
     If the business view is to execute your application:
          Read a copy of the VWPProgramInstance.tag template
          Search for and replace tokens with partner metadata (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 partner metadata (or defaults)
               Append the output to a target file
          End (for each parameter)

          If the business view is to be related to its VWP output target data:
               Read a copy of the BusinessViewVWPOutputTable.tag template
               Search for and replace tokens with partner metadata (or defaults)
               Append the output to a target file
          End (BV relation to its output)
     End (if BV to execute your application)

     If the business view is to be related to its input source data:
          Read a copy of the BusinessViewInputTable.tag template
          Search for and replace tokens with partner metadata (or defaults)
          Append the output to a target file
     End (BV relation to its source)
     If the business view is to be related to its output target data:
          Read a copy of the BusinessViewOutputTable.tag template
          Search for and replace tokens with partner metadata (or defaults)
          Append the output to a target file
     End (BV relation to its target)
End (for each business view)
```

Figure 11. Pseudocode for adding business views to the tag language file

The ISV_Sample program provides an example of adding business views and their source tables 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 business views

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

- You can specify that two business views run concurrently by defining a concurrent cascade relationship.
- You can specify that one business view starts after another business view successfully finishes processing by defining a post-processing cascade relationship.

To define the cascading business views:

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 cascade relationships.

Table 9. Templates for cascade relationships

Definition	Number of copies of template	Template to copy	Prerequisite template
Concurrent cascade relationship	One copy for each relationship	“ConcurrentCascade.tag” on page 82	“BusinessView.tag” on page 67
Post-processing cascade relationship	One copy for each relationship	“PostCascade.tag” on page 85	“BusinessView.tag” on page 67

Substituting values

Your program must supply the name of a business view and the name of another business view to:

- Start concurrently with the first business view
- Start after the first business view

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

Program logic

The following pseudocode examples show the logic that your program can use to build the portion of the tag file to relate business views.

Figure 12 shows how to relate two business views so that they run concurrently.

```
Read a copy of the ConcurrentCascade.tag template
Search for and replace tokens with partner metadata (or defaults)
Append the output to a target file
End (relate BV concurrent processing)
```

Figure 12. Concurrent cascade of two business views

Figure 13 shows how to relate two business views so that one starts the other when the first one finishes processing.

```
Read a copy of the PostCascade.tag template
Search for and replace tokens with partner metadata (or defaults)
Append the output to a target file
End (relate BVs post-processing)
```

Figure 13. Concurrent cascade of two business views

Importing metadata from the tag language file

You can import metadata from the tag language file by using a command-prompt interface or the user interface. This section describes how to use the command-prompt interface. For information about using the user interface, see the Visual Warehouse 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
```

where:

tag-filename
Is the full path and file name of the tag language file

log-pathname
Is the fully qualified path name of the log file

target-control-db
Is the name of the control database that is the target database for the import

userid
Is the user ID to use to access the control database

password
Is the password to use to access the control database

To get help for the import command’s 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 Visual Warehouse, see *Managing Visual Warehouse*.

Preparing the business views to run

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

1. Specify passwords for the following objects:
   - Any agent sites that you imported
   - Any information resources or warehouses (data sources and data targets) that you imported
2. If the source tables or files map directly to the target table, map the source columns to the target columns.
3. Define specific date and time schedules for the business views. You can also define cascade relationships if you did not do so in the tag language file.
4. Promote the business views to test status.
5. To test the business views, run them by selecting them in the Run Business View window.

   If you need to make changes:

   a. Demote the business views to development status if necessary
   b. Make the changes
   c. Promote the business views to test status again

   Be sure to update your program to account for these changes.
6. Promote the business views to production status to activate their schedules.

   Your business views will now run on an automated schedule.

Chapter 3. Exporting metadata from Visual Warehouse

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

To export metadata from Visual Warehouse:

- Select the objects for which to export metadata
- Export the metadata to a tag language file

Selecting objects for which to export metadata

Most Visual Warehouse objects are specific to Visual Warehouse. 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 data warehouse.

For example, one partner tool might unload data from a database into a target data file. Another partner tool might use the data 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 data file and load it into a target database. If you export the metadata for the databases and data files from Visual Warehouse, you can make sure that all the partner tools are using the same data definitions.

To define source databases, export one or more information resources (all tables and columns are included automatically). To define a target database, export a warehouse (along with its associated tables and columns).

When you export the objects, Visual Warehouse 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 8. Visual Warehouse metadata” on page 113. For the syntax and structure of a tag language file, see “Chapter 11. Tag language” on page 221 and “Chapter 12. What a tag language file should look like” on page 259.

Table 10 on page 36 shows the mapping between the logical Visual Warehouse objects and the tag language object that represents the logical object.

Table 10. Logical objects for source and target databases

Visual Warehouse logical object	Object in tag language file	Description	See:
Information resource	DATABASE	Source database or file	"DATABASE object" on page 113
Warehouse	DATABASE	Target database or file	"DATABASE object" on page 113
Table	TABLES	Table in source or target database	"TABLES object" on page 120
Column	COLUMN	Column in table or field in file	"COLUMN object" on page 124

Exporting metadata into a tag language file

You can use the Visual Warehouse user interface or a command-prompt interface to export metadata from Visual Warehouse. This section covers how to use the command-prompt interface. For information about using the user interface, see the Visual Warehouse online help and *Managing Visual Warehouse*.

First, you create an .INP file with the list of information resources and warehouses you want to export. For example:

```
<IR>
LOG_STAT_IR
LOG_STAT_REP
```

LOG_STAT_IR is an information resource, and LOG_STAT_REP is a warehouse. Visual Warehouse automatically exports the tables and columns that are associated with LOG_STAT_IR. To export the tables and columns that are associated with LOG_STAT_REP, you must export the business views that have a target table in LOG_STAT_REP. Add the business views to the .INP file. For example:

```
<BV>
SUCCESS_24HOUR
SUCCESS_ALL1
SUCCESSFUL_PULLS
```

In the tag language file, look for the target tables and ignore the rest of the business view.

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

```
iwh2exp2 INPfilename controlDBname userid password
```

where:

- *INPfilename* is the full path and file name of the .INP file.

  Create this file in a read / write directory because Visual Warehouse will write the tag language file in this directory. Visual Warehouse names the tag language file *INPfilename*.TAG.
- *controlDBname* is the name of the control database.
- *userID* is the user ID required to access the control database.
- *password* is the password that is required to access the control database.

For more information about exporting metadata from Visual Warehouse, see *Managing Visual Warehouse*.

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 Visual Warehouse to another. If you want to migrate Visual Warehouse, see *Planning and Installing Visual Warehouse and DataGuide*.

Chapter 4. Exporting metadata from DataGuide

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

To export metadata from DataGuide:

1. Select the types of metadata to export.
2. Export the metadata from DataGuide.

When you export metadata from DataGuide, you can generate tag language in two formats. For example:

- If you export using the DataGuide product windows or the FLGExport API, the tag language generated is in DataGuide tag language format. You can export metadata from the Windows 95 or Windows NT command line. See the Visual Warehouse README file for more information.
- If you export using the FLGMdisExport API, the tag language generated is in MDIS format.

Selecting metadata to export

The metadata that you can export from DataGuide 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.

You can export two types of object types from DataGuide:

Predefined object types,
Object types whose definitions are shipped with DataGuide. See "Chapter 10. DataGuide object types" on page 155 for a description of those object types.

You can only export predefined object types to an MDIS tag language file. See "Exporting MDIS-conforming tag language files" on page 42 for more information.

User-defined object types,
Object types that DataGuide administrators define.

User-defined object types

To obtain information about a user-defined object type, you must contact the administrator who defined the object type. However, all user-defined object types have certain components in common. These components must follow certain rules to be valid in DataGuide. Table 11 summarizes these components and rules.

Table 11. User-defined components and rules

Component	Rules
Object type name	• 80 character maximum. • It must not contain null characters. • It must not be all blank characters.
Object type short name	• 8 character (SBCS) maximum. • First character must be uppercase or lowercase English alphabetic, @, #, or $. • Subsequent characters must be uppercase or lowercase English alphanumeric, @, #, $, or _.
DataGuide object type common properties	For information about the DataGuide object type common properties, see Table 12.
User-defined properties	Each object type can have up to 255 properties. For information about the characteristics of properties, see Table 13 on page 41.
Universal unique identifier (UUI)	The UUI consists of up to five properties that uniquely identify the object type.

DataGuide defines five properties that are common to all DataGuide object types. These five properties are summarized in Table 12.

Table 12. DataGuide object type common properties

External name of property	Property short name	Definition
Object type identifier	OBJTYPID[1]	DataGuide generates this value, which uniquely identifies the object type of an object within the scope of the local information catalog.
Instance identifier	INSTIDNT[1]	DataGuide generates this value, which uniquely identifies an object within the scope of the local information catalog.
Name	NAME	You provide the name of an object. Choose names that users readily recognize and understand.

Table 12. DataGuide object type common properties (continued)

External name of property	Property short name	Definition
Last Changed Date and Time	UPDATIME[1]	DataGuide generates this value, indicating the date and time that the object was last changed.
Last Changed By	UPDATEBY[1]	DataGuide generates this value, showing the user ID of the DataGuide session that last updated the Last Changed Date and Time property.

1. If you selected the **Hide system generated properties** check box in the DataGuide Settings notebook, you won't see this property in the object's description.

All user-defined properties have certain components in common. These components must follow certain rules to be valid in DataGuide. Table 13 summarizes these components and rules.

Table 13. Rules for user-defined properties

Component	Rules
Property name	• 80 character maximum. • It must not contain null characters. • It must not be all blank characters.
Property short name	• 8 character (SBCS) maximum. • First character must be uppercase or lowercase English alphabetic, @, #, or $. • Subsequent characters must be uppercase or lowercase English alphanumeric, @, #, $, or _. • It must not be an SQL reserved word. • It must be unique; if you type a name that already exists in this object type, DataGuide asks you for another name.

Table 13. Rules for user-defined properties (continued)

Component	Rules
Data type	**CHAR** Up to 254 characters **VARCHAR** Up to 4 000 characters **LONG VARCHAR** Up to 32 700 characters **TIMESTAMP** Exactly 26 characters, in this format: yyyy-mm-dd-hh.mm.ss.nnnnnn: An object type can have up to 14 LONG VARCHAR properties.
Size	The size must be within the range for the data type that you selected.
Entry required	Whether entry of this property is required whenever an object of this type is created

For more information about user-defined object types, see *Managing DataGuide*.

The following section contains instructions on exporting an MDIS-conforming tag language file from the command line. For information about using the DataGuide product windows to export a DataGuide tag language file, see *Managing DataGuide*.

Exporting MDIS-conforming 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 DataGuide for the first time, you must merge the information in the DataGuide 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 DataGuide.)

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 DataGuide.)

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.
- Each keyword must be preceded by either a / or - character.
- 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 (the line break in this example is not significant, you can continue typing until the text wraps to the next line):

```
DGUIDE /USERID longods /PASSWORD secret /DGNAME DGV5SAMP /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 don’t specify this optional keyword for the DGUIDE command, you are logged on as a user. You can export metadata as a user, however, you cannot perform all administrator tasks.

/DGNAME

Your DataGuide information catalog’s 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 DGV5SAMP
```

/LOGFILE

Specifies the file destination for messages that DataGuide generates during MDIS import or MDIS export.

Unless you specify a full drive, path, and file name, DataGuide 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, DataGuide 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 is not limited to containing MDIS metadata, but /MDIS_EXPORT only exports 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 the previous list, the parts of the name are represented with their MDIS name. You can export only a subset of DataGuide object types to the MDIS tag language. To find the equivalent DataGuide names for objects types that you can export, see the DataGuide object type property tables that begin on page61. For each object type that conforms to the Metadata Interchange Specification (MDIS), the MDIS equivalent for each property appears in the column entitled **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 DataGuide names in the **Property name** and **Property short name** columns.

For each part, enter the value of the named property for the object you want to export. You can use an * as a wildcard within, or instead of, any of the parts. If you enter nothing for a part, DataGuide uses the not-applicable symbol when searching for objects to export. (The not-applicable symbol is a hyphen unless you identified a different symbol when you created the information catalog.)

Examples (the line break within the first example is not significant, you can continue typing until the text wraps to the next line):

```
/MDIS_EXPORT d:\tagfile.tag /OBJTYPE record /OBJECTS
    stl11ffh.Cobol Files.labriejj.CCD-REC
/MDIS_EXPORT d:\tagfile.tag /OBJTYPE subschema /OBJECTS STLS1W71.CELDIAL.*.
/MDIS_EXPORT d:\tagfile.tag /OBJTYPE dimension /OBJECTS *.STLS3W71..Market
```

The equivalent MDIS name for each DataGuide property short name is shown in “Predefined DataGuide object types” on page 159.

Limitation for exporting objects of type "Multi-dimensional database": DataGuide exports only the first level of member information, not the entire multi-dimensional model.

/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 for DB2 for AIX, DB2 PE, DB2 for Windows NT, and DB2 for Windows 95 databases are case sensitive; you must type them exactly as specified.

/TRACE

The level of trace information to send to the DataGuide 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 DataGuide problems.

0 The default level. Level includes all messages and warning, error, and severe error conditions.

1 Includes entry and exit records of the highest level DataGuide functions.

2 Includes extremely granular entry and exit records of the DataGuide functions.

3 Includes input and output parameters (excluding input or output structures).

4 Includes all input or output structures that are passed to and used by DataGuide.

/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, OS/400, AIX, or MVS TSO user ID.

Example:

```
/USERID longods
```

Chapter 5. Importing metadata into DataGuide

You can import metadata from Visual Warehouse and partner applications to provide information about the data in a warehouse for the end users of the warehouse. You can exchange metadata with partner applications that also provide some cataloging facilities.

To import metadata into DataGuide:

1. Select the types of metadata to import.
2. Import the metadata into DataGuide.

Selecting metadata to import

The metadata that you can export from DataGuide 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.

When you import metadata into DataGuide, you can import the tag language in two formats:

- A format that is used by both DataGuide and Visual Warehouse
- A format that conforms to MDIS.

Predefined object types

DataGuide includes predefined object types that can be exchanged with metadata from other Visual Warehouse components and other MDIS-conforming products from IBM and other companies. MDIS data maps only to the predefined object types. For a description of the predefined objects and object properties, see “Chapter 10. DataGuide object types” on page 155.

User-defined object types

If you need an object type that is not already defined in DataGuide, you can create your own object types. You cannot import

When you create your own object types, start by creating a prototype for each object type that you need. Then create one or two sample objects (see *Managing DataGuide* for information about objects). Check how the objects appear in a Description View, especially the order in which the properties are listed. Try entering different values for each property to be sure you have the right data types and sizes. You might want to consult with your database administrator and some of your users to ensure that the properties that you specify meet your work group's needs.

If you aren't satisfied with your prototype, you can easily delete it and your sample objects and start again. After you create an object type, the only way to change or delete its properties is to delete the object type *and all objects of that type* and create a new object type.

Consider how many object types you will need. DataGuide limits the number of object types that you can create in an information catalog to 999 999, and the number of objects you can create for each type to 99 999 999. This limit includes all the object types ever created, even the ones that were deleted.

You can create an object type using the DataGuide windows, tag language, or Application Programming Interface (API). See the *DataGuide Programming Guide and Reference* for information on using DataGuide APIs.

The following sections describe how to create, update, and delete an object type using tag language. For information about creating, updating, and deleting an object type using the DataGuide windows, see *Managing DataGuide*. See the *DataGuide Programming Guide and Reference* for information on using DataGuide APIs to create, update and delete object types.

Creating an object type using DataGuide tag language

1. Enter the following lines in your tag language file:

   ```
   :ACTION.OBJTYPE(ADD)
   :OBJECT.TYPE(short_name_of_object_type)
        PHYNAME (name_of_table)
        CATEGORY(category_of_object_type)
        EXTNAME(external_name_of_object_type)
        ICOFILE(name_of_OS/2_icon_file)
        ICWFILE(name_of_Windows_icon_file)
   ```

   After each keyword, type an appropriate value within the parentheses:

   TYPE The short name of the object type. The rules for short names are:

   - 8 character (SBCS) maximum.
   - First character must be uppercase or lowercase English alphabetic, @, #, or $.
   - Subsequent characters must be uppercase or lowercase English alphanumeric, @, #, $, or _.
   - It must be unique to the information catalog.

   PHYNAME
   The name of a DB2 table where DataGuide stores objects of this type.

If your DB2 tables follow naming conventions, you can use PHYNAME to give the underlying tables in your information catalog a different name from the object type name.

If you don’t specify this property, DataGuide uses the short name that you gave as the TYPE keyword.

You can add the PHYNAME keyword only if you use a tag language file to create the object type. You can’t add it through the DataGuide product windows.

CATEGORY
GROUPING, ELEMENTAL, CONTACT, DICTIONARY, or SUPPORT.

EXTNAME
The external name of the object type. The rules for external names are:

- 80 character maximum.
- It must not contain null characters.
- It must not be all blank characters.

ICOFILE
The name of the OS/2 icon file, including its extension. You give the drive and path information where the icon file exists as part of the IMPORT command when you import your tag language file.

ICWFILE
The name of the Windows icon file, including its extension. You give the drive and path information where the icon file exists as part of the IMPORT command when you import your tag language file.

2. Type lines for each property you want to give your object type:

```
:PROPERTY.SHRTNAME(short_name) DT(data_type) DL(size)
     UUISEQ(position_in_UUI) NULLS(y_or_n) EXTNAME(property_name)
```

SHRTNAME
The property short name. The rules for property short names are:

- 8 character (SBCS) maximum.
- The first character must be uppercase or lowercase English alphabetic, @, #, or $.
- Subsequent characters must be uppercase or lowercase English alphanumeric, @, #, $, or _.
- It must not be an SQL reserved word.
- It must be unique; if you type a name that already exists in this object type, DataGuide asks you for another name.

DT The data type: **C**, **V**, **L**, or **T**.

C (CHAR)
Up to 254 characters

V (VARCHAR)
Up to 4 000 characters

L (LONG VARCHAR)
Up to 32 700 characters

T (TIMESTAMP)
26 characters, in this format:

yyyy-mm-dd-hh.mm.ss.nnnnnn

DL The size for the property.

UUISEQ
The position that this property has in the UUI: **1**, **2**, **3**, **4**, or **5**. Include this keyword only if you want the property to be part of the UUI.

NULLS
Whether entry is required:

N Entry of a value is required

Y Entry of a value is *not* required

EXTNAME
The property name. The rules for property names are:

- 80 character maximum.
- It must not contain null characters.
- It must not be all blank characters.

If you want to make the NAME property part of the UUI for this object type, the only keywords that you can use for the property are SHRTNAME and UUISEQ. DataGuide defines values for other keywords, so you don't specify them or their values here.

After adding all properties for your object type, your tag language file looks something like Figure 14 on page 51. Figure 14 on page 51 shows an abbreviated version of the Text-based reports object type, which is one of the predefined object types provided with DataGuide. The complete object type definition is available in the X:\VWSWIN\DGWIN\TYPES directory, where X is the drive where Visual Warehouse is installed.

```
:COMMENT.------------------------------------------------------------
:Generating the report object definitions.
:COMMENT.------------------------------------------------------------
:ACTION.OBJTYPE(MERGE)
:OBJECT.TYPE(REPORT) CATEGORY(ELEMENTAL) PHYNAME(REPORTS)
    EXTNAME(Text based reports) ICWFILE(flgnyrep.ico)
:PROPERTY. SHRTNAME(NAME)                          UUISEQ(0)
:PROPERTY. SHRTNAME(SHRTDESC)   DT(V)   DL(250)    UUISEQ(0)   NULLS(Y)
    EXTNAME(Short description)
:PROPERTY. SHRTNAME(LONGDESC)   DT(L)   DL(32700)  UUISEQ(0)   NULLS(Y)
    EXTNAME(Long description)
:PROPERTY. SHRTNAME(ACTIONS)    DT(V)   DL(254)    UUISEQ(0)   NULLS(Y)
    EXTNAME(Actions)
:PROPERTY. SHRTNAME(TITLE)      DT(V)   DL(254)    UUISEQ(0)   NULLS(N)
    EXTNAME(Report title)
:PROPERTY. SHRTNAME(RPRTDATE)   DT(C)   DL(26)     UUISEQ(0)   NULLS(Y)
    EXTNAME(Report publication date)
:PROPERTY. SHRTNAME(RPRTFRMT)   DT(V)   DL(80)     UUISEQ(0)   NULLS(Y)
    EXTNAME(Report presentation format)
:PROPERTY. SHRTNAME(DBPRESNT)   DT(V)   DL(254)    UUISEQ(0)   NULLS(Y)
    EXTNAME(Report presentation requirements)
:PROPERTY. SHRTNAME(OWNER)      DT(V)   DL(80)     UUISEQ(0)   NULLS(Y)
    EXTNAME(Report owner)
:PROPERTY. SHRTNAME(FILENAME)   DT(V)   DL(254)    UUISEQ(1)   NULLS(N)
    EXTNAME(Report filename)
:PROPERTY. SHRTNAME(TYPE)       DT(V)   DL(80)     UUISEQ(2)   NULLS(N)
    EXTNAME(Report class or type)
:PROPERTY. SHRTNAME(URL)        DT(V)   DL(254)    UUISEQ(0)   NULLS(Y)
    EXTNAME(URL to access data)
:COMMIT.CHKPID(31)
```

Figure 14. Sample tag language file for an object type

Updating an object type using DataGuide tag language

1. Enter the following lines in your tag language file:

   ```
   :ACTION.OBJTYPE(UPDATE)
   :OBJECT.TYPE(short_name_of_object_type)
   ```

2. To change the external name, add the following line:

   ```
        EXTNAME(new_external_name_of_object_type)
   ```

3. To change the object type’s icon, add the following line depending on your operating system:

   ```
        ICOFILE(new_OS/2_icon_filename)
        ICWFILE(new_Windows_icon_filename)
   ```

   After each keyword, type an appropriate value within the parentheses:

   TYPE The short name of the object type that you are updating.

EXTNAME
The new external name of the object type. The rules for external names are:

- 80 character maximum.
- It must not contain null characters.
- It must not be all blank characters.

ICOFILE
The name of the new OS/2 icon file, including its extension. You give the drive and path information where the icon file exists as part of the IMPORT command when you import your tag language file.

ICWFILE
The name of the new Windows icon file, including its extension. You give the drive and path information where the icon file exists as part of the IMPORT command when you import your tag language file.

4. To add an optional property, enter the following lines in your tag language file:

```
:ACTION.OBJTYPE(APPEND)
:OBJECT.TYPE(short_name_of_object_type)
:PROPERTY.SHRTNAME(short_name_of_new_property) DT(data_type) DL(size)
     UUISEQ(0) NULLS(y) EXTNAME(external_name_of_new_property)
```

After each keyword, type an appropriate value within the parentheses.

Any property that you add to an object type after you create it must be an optional property, so the value for UUISEQ must be 0 and NULLS must be Y.

TYPE The short name of the object type that you are updating.

SHRTNAME
The property short name. The rules for property short names are:

- 8 character (SBCS) maximum.
- First character must be uppercase or lowercase English alphabetic, @, #, or $.
- Subsequent characters must be uppercase or lowercase English alphanumeric, @, #, $, or _.
- It must not be an SQL reserved word.
- It must be unique; if you type a name that already exists in this object type, DataGuide asks you for another name.

DT The data type: **C**, **V**, **L**, or **T**.

C (CHAR)
Up to 254 characters

V (VARCHAR)
Up to 4 000 characters

L (LONG VARCHAR)
Up to 32 700 characters

T (TIMESTAMP)
Exactly 26 characters, in this format:

yyyy-mm-dd-hh.mm.ss.nnnnnn

DL The size for the property.

EXTNAME
The external name of the property. The rules for property names are:

- 80 character maximum.
- It must not contain null characters.
- It must not be all blank characters.

Deleting an object type using DataGuide tag language

To delete an object type and any underlying object types, enter the following lines in your tag language file:

```
:ACTION.OBJTYPE(DELETE_EXT)
:OBJECT.TYPE(short_name_of_object_type)
```

After each keyword, type an appropriate value within the parentheses:

TYPE The short name of the object type that you are deleting.

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 DataGuide and Visual Warehouse. See “Chapter 10. DataGuide object types” on page 155 for mappings of DataGuide 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 MDIS-conforming tag language files” on page 42.

If you want to convert MDIS tag language into a DataGuide tag language file, see *Managing DataGuide*.

Importing MDIS-conforming tag language files

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

- All the parts, except where specified, are case insensitive.
- Each keyword must be preceded by either a / or - character.
- 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 (the line break in this example is not significant, continue typing until the text wraps to the next line):

```
DGUIDE /USERID longods /PASSWORD secret /DGNAME DGV5SAMP /ADMIN
/MDIS_IMPORT c:\mdis.tag /LOGFILE c:\mdis.log
```

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

/DGNAME
Your DataGuide information catalog’s 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 DGV5SAMP
```

/LOGFILE
This parameter is required.

Specifies the file destination for messages that DataGuide generates during MDIS import or MDIS export. Unless you specify a full drive, path, and file name, DataGuide 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, DataGuide 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 for DB2 for AIX, DB2 PE, DB2 for Windows NT, and DB2 for Windows 95 databases are case sensitive; you must type them exactly as specified.

/TRACE

The level of trace information to send to the DataGuide 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 DataGuide problems.

0 The default. Includes all messages and warning, error, and severe error conditions.

1 Includes entry and exit records of the highest level DataGuide functions.

2 Includes extremely granular entry and exit records of the DataGuide functions.

3 Includes input and output parameters (excluding input or output structures).

4 Includes all input or output structures that are passed to and used by DataGuide.

/USERID

Your DataGuide 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, OS/400, AIX, or MVS TSO user ID.

Example:

```
/USERID longods
```

Importing DataGuide tag language files

To open a DataGuide information catalog from a DOS command prompt, enter the DGUIDE command. Adhere to the following rules for the command syntax:

- All the parts, except where specified, are case insensitive.
- Each keyword must be preceded by either a / or - character.
- All keywords that follow the DGUIDE command on the same line are required. All keywords that follow /IMPORT on the same line are required if you choose to use /IMPORT.
- Underlined choices are defaults.

```
DGUIDE /USERID userid /PASSWORD password /DGNAME dgname /IMPORT filename
/LOGFILE filename /ADMIN
```

Optional keywords:

```
/TRACE 0|1|2|3|4
 /RESTART B|C
```

Optional import keyword:

```
/ICOPATH iconpath
```

For example, to open the sample DataGuide information catalog as an administrator, type:

```
DGUIDE /USERID longods /PASSWORD secret /DGNAME DGV5SAMP /ADMIN
```

/ADMIN
Specifies that you are logging on as an administrator. If you don’t specify this optional keyword for the DGUIDE command, you cannot perform administrator tasks.

/DGNAME
Your DataGuide information catalog’s name.

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

Example:

```
/DGNAME DGV5SAMP
```

/ICOPATH
Valid only with /IMPORT; optional.

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

Example:

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

/IMPORT

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

Example:

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

This keyword bypasses the DataGuide user interface and performs the import function as a batch command.

/LOGFILE

Valid only with /IMPORT; required with /IMPORT.

Specifies the file destination for messages that DataGuide generates during import. Unless you specify a full drive, path, and file name, DataGuide 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 for DB2 for AIX, DB2 PE, DB2 UDB EEE, DB2 for Windows NT, and DB2 for Windows 95 databases are case sensitive; you must type them exactly as specified.

/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 Default. Imports the tag language file from the last point at which DataGuide successfully committed changes to the information catalog.

/TRACE

The level of trace information to be sent to the DataGuide 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 have to specify a higher level if you call the IBM Support Center to diagnose DataGuide problems.

0 Default. Includes all messages and warning, error, and severe error conditions.

1 Includes entry and exit records of the highest level DataGuide functions.

2 Includes extremely granular entry and exit records of the DataGuide 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 DataGuide.

/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, OS/400, AIX, or MVS TSO user ID.

Example:

```
/USERID longods
```

Chapter 6. Ensuring users can launch programs from DataGuide

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

Ensure that:

- Your users have the appropriate application software installed on their workstations or LAN.
- Your users' workstation PATH environment variables include the directory paths where the programs have been installed.
- Your users have the necessary authorization to the databases or file systems where the information they need is stored.
- The Programs objects in the information catalog include the correct invocation syntax for the operating systems on which your users will run the programs.

Additional considerations for DataGuide for the Web users

In addition to the previous list, there are specific considerations to be aware of when you set up the Web environment so that DataGuide for the Web users can launch programs.

Ensure that:

- The application program users want to launch is accessible to the Web server. For example, the DataGuide sample data file is located in a directory on the Web server.
- The Web client has the program users want to launch installed.

  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, you do not need to have the application installed, it can be accessed directly through the Web browser.
- The required MIME types are identified in the Web server configuration file (http.cnf) for the application program users will launch. There should be an AddType directive with the file extension of the program users want to launch. For example, if users want to launch a Lotus 1-2-3 spreadsheet with a file type of .WK4, then the AddType directive should be defined as:

```
AddType .WK4 application/x-lotus123 binary
```

If users are using a Web server other than Lotus Go, the MIME types are defined differently. See your Web server documentation.

- For some versions of Netscape Navigator, helpers are set up to recognize file types and start the corresponding application program. Microsoft Internet Explorer does not user helpers. Instead, Internet Explorer uses the file type and program associations used by Windows Explorer; no set up is required for Internet Explorer to recognize a file type.
- The DataGuide object from which users want to launch the program has the **URL to access data** property defined. The value for the property is a link to directly launch the program.

To launch a program from a DataGuide for the Web object:

1. In the list pane, click on the object from which you want to launch 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 URL for the object is launched.

Part 2. Metadata reference

Chapter 7. Metadata templates

Refer to this chapter for detailed information about each template that is provided with 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.

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 14 lists the metadata templates that are supplied with Visual Warehouse and the section that covers each template.

Table 14. Metadata templates supplied with Visual Warehouse

Template	**Description**	**See:**
AgentSite.tag	Defines an agent site from which the agent accesses the data source or target warehouse, or on which a Visual Warehouse program runs.	"AgentSite.tag" on page 65
BusinessView.tag	Defines a business view that is to be managed by Visual Warehouse.	"BusinessView.tag" on page 67
BusinessViewInputTable.tag	Specifies that a business view use a given source table.	"BusinessViewInputTable.tag" on page 72
BusinessViewOutputTable.tag	Specifies that a business view uses a given target table.	"BusinessViewOutputTable.tag" on page 74

Table 14. Metadata templates supplied with Visual Warehouse (continued)

Template	Description	See:
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 75
Column.tag	Defines a column or field in a table, segment, or file.	“Column.tag” on page 77
ConcurrentCascade.tag	Indicates that two business views are to be started at the same time.	“ConcurrentCascade.tag” on page 82
HeaderInfo.tag	Declares all the object type definitions needed by Visual Warehouse to declare a tag language file.	“HeaderInfo.tag” on page 84
SourceDataBase.tag	Defines data sources to import into Visual Warehouse.	“SourceDataBase.tag” on page 86
SubjectArea.tag	Defines a subject area to contain the business views being created.	“SubjectArea.tag” on page 89
Table.tag	Defines a table or file that Visual Warehouse is to access.	“Table.tag” on page 91
VWPGroup.tag	Defines a group that is to contain any Visual Warehouse program being defined.	“VWPGroup.tag” on page 95
VWPProgramInstance	Modifies the definition of a Visual Warehouse program for use by a specific business view.	“VWPProgramInstance.tag” on page 97
VWPProgramInstanceParameter.tag	Adds or modifies a parameter that Visual Warehouse passes to an instance of a Visual Warehouse program used by a specific business view.	“VWPProgramInstanceParameter.tag” on page 99
VWPProgramTemplate.tag	Defines a Visual Warehouse program.	“VWPProgramTemplate.tag” on page 101

Table 14. Metadata templates supplied with Visual Warehouse (continued)

Template	Description	See:
VWPProgramTemplateParameter.tag	Defines a parameter that Visual Warehouse is to pass to a Visual Warehouse program.	“VWPProgramTemplateParameter.tag” on page 104
WarehouseDataBase.tag	Defines target warehouses to import into Visual Warehouse.	“WarehouseDataBase.tag” on page 107

AgentSite.tag

Use this template to define an agent site:

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

You can use one of the following agent site types:

- An agent site that is already defined in the Visual 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.

AgentSite.tag

Tokens

Table 15 provides information about each token in the template.

Table 15. AgentSite.tag tokens

Token	Description	Allowed values	Window or notebook: field
		Entity parameters	
**AgentSite*	The business name of a new agent site or the default agent site. If you specify a new name, it must be unique within the Visual 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.	Agent site: Name
**AgentSiteDescription*	The short description of the agent site. This token is optional.	A text string, up to 200 bytes in length.	Agent site: Description
**AgentSiteNotes*	The long description of the agent site. This token is optional.	A text string, up to 32700 bytes in length.	Agent site: Notes
**AgentSiteOSType*	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	Agent site: User ID

Table 15. AgentSite.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**AgentSiteTCP/IPHostname*	TCP/IP host name of the agent site. This token is required.	A text string, up to 200 bytes in length.	Agent site: Host Name
**AgentSiteUserid*	User ID under which the agent runs. This token is required.	A text string, up to 36 bytes in length.	Agent site: User ID
	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.	None

Examples of values

Table 16 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 16. Example values for AgentSite.tag tokens

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

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 17 provides information about each token in the template.

Table 17. 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
**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

Table 17. BusinessView.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**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
**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

Table 17. BusinessView.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**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
**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 fail if an SQL warning code is issued. **ISV_BVSQLWarning_Error** The business view is to process with a warning if an SQL warning code is issued.	Business View: SQL Warning Processing Options
Relationship parameters			

Table 17. BusinessView.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**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
**CurrentCheckPointID++*	An 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 18 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 18. 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

Table 18. Example values for BusinessView.tag tokens (continued)

Token	Example value
**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_Uneditioned_Append).

Tokens

Table 19 on page 73 provides information about each token in the template.

Table 19. BusinessViewInputTable.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	The 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*	The 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*	The 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
**TablePhysicalName*	The physical table name as defined to the DBMS or file system. This token is required.	A text string, up to 80 bytes in length.	Table: Name Business View: Database Table Name
**CurrentCheckPointID++*	An 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 20 on page 74 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 20. 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_Uneditioned_Append).

Tokens

Table 21 provides information about each token in the template.

Table 21. BusinessViewOutputTable.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	The name of the business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**DatabaseName*	The name of the database that contains the table. This token is required.	A text string, up to 80 bytes in length.	Information resource: Database

Table 21. BusinessViewOutputTable.tag tokens (continued). This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**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.	Table: Name Business View: Table Name Qualifier
**TablePhysicalName*	The physical table name as defined to the DBMS or file system. This token is required.	A text string, up to 80 bytes in length.	Table: Name Business View: Database Table Name
**CurrentCheckPointID++*	An 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 22 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 22. Example values for BusinessViewOutputTable.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_7
**DatabaseName*	Finance Warehouse
**TableOwner*	DB2ADMIN
**TablePhysicalName*	GEOGRAPHY
**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.

BusinessViewVWPOutputTable.tag

Tokens

Table 23 provides information about each token in the template.

Table 23. BusinessViewVWPOutputTable.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	The name of the business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**DatabaseName*	The 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*	The 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*	The physical table name as defined to the DBMS or file system. This token is required.	A text string, up to 80 bytes in length.	Table: Name Business View: Database Table Name
**CurrentCheckPointID++*	An 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 24 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 24. Example values for VWPOutputTable.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_7
**DatabaseName*	Finance Warehouse

Table 24. Example values for VWPOutputTable.tag tokens (continued)

Token	Example value
**TableOwner*	DB2ADMIN
**TablePhysicalName*	GEOGRAPHY
**CurrentCheckPointID++*	15

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 data sources and data 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 data sources or data targets by using the Table.tag template (see page 91).

Tokens

Table 25 provides information about each token in the template.

Table 25. Column.tag tokens

Token	Description	Allowed values	Window or notebook: field
	Entity parameters		
**ColumnName*	The name of the column or field. The name must be unique within the table. This token is required.	A text string, up to 80 bytes in length.	Table: (Column) Name
**ColumnDescription*	The short description of the column or field. This token is optional.	A text string, up to 200 bytes in length.	Table: (Column) Description
**ColumnNotes*	The long description of the column or field. This token is optional.	A text string, up to 32700 bytes in length.	Table: (Column) Notes

Table 25. Column.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**ColumnUserActions*	Actions that a user can perform on this column or field. This token is optional.	A text string, up to 254 bytes in length.	None
**ColumnLength*	The length of the column or field being created. This token is required.	A numeric value.	Table: Length / Precision
**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.	Table: Length / Precision
**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.	None
**ColumnPositionNumber*	A number, starting with 0, that indicates the order of the column within the row. This token is required.	A numeric value.	Table: Column Information
**ColumnAllowsNulls*	Flag indicating 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.	Table: Column Information

Table 25. Column.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**ColumnDataIsText*	Flag indicating whether the column or field contains only text data. 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.	Table: Column Information
**ColumnNativeDataType*	The data type of the column or field as defined to the DBMS 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: Native Type

Table 25. Column.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**ColumnNativeDataType* (continued)	The data type of the column or field as defined to the DBMS 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_DATETIMEYEARTOFRACTION • ISV_NATIVE_DBCLOB • ISV_NATIVE_BIGINT	Table: Native Type
		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.	None
**DatabasePhysicalName*	The physical name of the database or file that contains the column or field, as defined to the DBMS or file system. This token is required.	A text string, up to 40 bytes in length.	Information resource: Database Warehouse: Database

Table 25. Column.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**TablePhysicalName*	The physical name of the table or file that contains the column as defined to the DBMS or file system. This token is required.	A text string, up to 80 bytes in length.	None
**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.	None
**CurrentCheckPointID++*	An 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 26 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 26. 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
**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

Table 26. Example values for Column.tag tokens (continued)

Token	Example value
**TableOwner*	DB2ADMIN
**TablePhysicalName*	GEOGRAPHY
**CurrentCheckPointID++*	8

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 27 provides information about each token in the template.

Table 27. ConcurrentCascade.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	The name of the business view. This token is required.	A text string, up to 80 bytes in length.	Business View: Business Name
**ConcurrentBVName*	The 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++*	An 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 28 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 28. Example values for ConcurrentCascade.tag tokens

Token	Example value
**BVName*	Revenue_by_Geography_7
**ConcurrentBVName*	Revenue_by_Geography_6
**CurrentCheckPointID++*	16

HeaderInfo.tag

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

Tokens

Table 29 provides information about each token in the template.

Table 29. HeaderInfo.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**SecurityGroup*	The security group that is to contain all the objects you are importing. This token is required, and you must specify the default security group.	`ISV_DEFAULTSECURITYGROUP` for the default security group.	Groups: Name
**CurrentCheckPointID++*	An 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 30 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 30. Example values for Header.tag tokens

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

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 31 provides information about each token in the template.

Table 31. PostCascade.tag tokens. This template contains only relationship parameters.

Token	Description	Allowed values	Window or notebook: field
**BVName*	The 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*	The 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++*	An 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 32 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 32. Example values for PostCascade.tag tokens

Token	Example value
**BVName*	Revenue by geography 7
**PostBVName*	Revenue for all geographies
**CurrentCheckPointID++*	17

SourceDataBase.tag

Use this template to define source databases, file systems, or files to import into Visual Warehouse. 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 33 provides information about each token in the template.

Table 33. SourceDataBase.tag tokens

Token	Description	Allowed values	Window or notebook: field
		Entity parameters	
**DatabaseName*	The name of the database The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Information resource: Name
**DatabaseDescription*	The short description of the database. This token is optional.	A text string, up to 200 bytes in length.	Information resource: Description
**DatabaseNotes*	The long description of the database. This token is optional.	A text string, up to 32700 bytes in length.	Information resource: Notes
**DatabaseContact*	The person to contact for information about this database. This token is optional.	A text string, up to 64 bytes in length.	Information resource: Contact

Table 33. SourceDataBase.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**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.	None
**DatabasePhysicalName*	The physical database name of the database as defined to the DBMS. This token is required.	A text string, up to 40 bytes in length.	Information resource: Database
**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	Resource Type

Table 33. SourceDataBase.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
Token	**Description**	**Allowed values**	**Window or notebook: field**
**DatabaseTypeExtended*	The type of AS/400 machine or file. This token is required.	One of the following values: **ISV_IR_DB2400CISC** DB2 for 400 for CISC machines **ISV_IR_DB2400RISC** DB2 for 400 for RISC machines **ISV_IR_FFLanLocalCmd** Local flat file **ISV_IR_FFLanFTPCopy** Local flat file sent using FTP from a remote system	Information resource: Type
**DatabaseUserid*	The user ID with which to access the database. This token is optional.	A text string, up to 36 bytes in length.	Information resource: User ID
		Relationship parameters	
**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.	Information resource: Selected Security Groups
**AgentSite*	The agent site to use for the source or target. 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.	Information resource: Selected Agent Sites
**CurrentCheckPointID++*	An 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 34 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 34. 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
**DatabasePhysicalName*	FINANCE
**DatabaseType*	ISV_IR_DB2Family
**DatabaseTypeExtended*	ISV_DEFAULTVALUE
**DatabaseUserid*	DB2ADMIN
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**AgentSite*	My agent site
**CurrentCheckPointID++*	5

SubjectArea.tag

Use this template to define a subject to contain the business views you are creating. Each tag language file must have at least one subject area to contain any business views you are creating. This template is required if you are defining business views.

This template also defines the relationship between the subject area and the security group that the header file specifies (see “HeaderInfo.tag” on page 84).

Tokens

Table 35 on page 90 provides information about each token in the template.

SubjectArea.tag

Table 35. SubjectArea.tag tokens

Token	Description	Allowed values	Window or notebook: field
		Entity parameters	
**SubjectArea*	The name of a group that is to contain all of the business views being created or being added to a particular subject area. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Subject: Name
**SubjectAreaDescription*	Short description of the group of business views. This token is optional.	A text string, up to 200 bytes in length.	Subject: Description
**SubjectAreaNotes*	Long description of the group of business views. This token is optional.	A text string, up to 32700 bytes in length.	Subject: Notes
		Relationship parameters	
**SecurityGroup*	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.	Subject: Selected 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.	None

Examples of values

Table 36 on page 91 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 36. Example values for SubjectArea.tag tokens

Token	Example value
**SubjectArea*	Group of business views generated for the partner tool
**SubjectAreaDescription*	This subject area contains all the business views generated for VW by the partner tool.
**SubjectAreaNotes*	The business views in this subject will ...
**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 Visual Warehouse is to access. You can use this template to define both source and target tables, files, and segments.

The template defines all the metadata that Visual Warehouse 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 37 provides information about each token in the template.

Table 37. Table.tag tokens

Token	Description	Allowed values	Window or notebook: field
		Entity parameters	

Table 37. Table.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**TableFullName*	The fully qualified name of a relational table or a file. This name is the concatenation of the value of the **TableOwner* and **TablePhysicalName* tokens, separated by a period. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Table: Name
**TableDescription*	The short description of the table. This token is optional.	A text string, up to 200 bytes in length.	Table: Description
**TableNotes*	The long description of the table. This token is optional.	A text string, up to 32700 bytes in length.	Table: Notes
**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.	Table: Name Business View: Table Name Qualifier
**TablePhysicalName*	The physical table name as defined to the DBMS or file system. This token is required.	A text string, up to 80 bytes in length.	Table: Name Business View: Database Table Name

Table 37. Table.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**TableBinaryIfFile*	A flag indicating 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.	File: File Transfer Type
**TableFirstRowNamesIfFile*	A flag indicating 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_CONTAINS_NAMES** The first row of the file contains data.	File: First Row Contains Column Names
**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*.	File: File Type

Table 37. Table.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**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.	File: Field Delimiter Character
	Relationship parameters		
**DatabaseName*	The name of the database that contains the table. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Information resource: Name Warehouse: Target Warehouse Name
**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.	Information resource: Database Warehouse: Database
**CurrentCheckPointID++*	An 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 38 on page 95 provides examples of values for each token to illustrate the kind of metadata you might provide for each token.

Table 38. 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
**DatabaseName*	Finance warehouse
**DatabasePhysicalName*	FINANCE
**CurrentCheckPointID++*	7

VWPGroup.tag

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

Tokens

Table 39 provides information about each token in the template.

Table 39. VWPGroup.tag tokens

Token	Description	Allowed values	Window or notebook: field
		Entity parameters	

Table 39. VWPGroup.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**VWPGroup*	The name of a group that is to contain all of the Visual Warehouse programs being created. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Program: Program Group
**VWPGroupDescription*	The short description of the group of Visual Warehouse programs. This token is optional.	A text string, up to 200 bytes in length.	None
**VWPGroupNotes*	The long description of the group of Visual Warehouse programs. This token is optional.	A text string, up to 32700 bytes in length.	None
	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.	None

Examples of values

Table 40 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 40. 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 VW for the partner tool
**VWPGroupNotes*	These programs can be used to ....
**CurrentCheckPointID++*	2

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 101). 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 41 provides information about each token in the template.

Table 41. VWPProgramInstance.tag tokens

Token	Description	Allowed values	Window or notebook: field
	Entity parameters		
**VWPInstanceNotes*	The 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
**VWPProgramInstanceKey*	A 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		

Table 41. VWPProgramInstance.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**BVName*	The name of the business view. This token is required.		Business View: Business Name
**VWPProgramTemplateName*	The 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++*	An 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 42 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 42. Example values for VWPProgramInstance.tag tokens

Token	Example value
**VWPInstanceNotes*	This program exports data from the Geography database.
**VWPProgramInstanceKey*	070000
**BVName*	Revenue by geography
**VWPProgramTemplateName*	My partner program
**CurrentCheckPointID++*	11

VWPProgramInstanceParameter.tag

Use this template to add or change a parameter that Visual Warehouse passes to an instance of a Visual Warehouse program for a specific business view. For example, you set a default value for a host name parameter in the VWPProgramTemplateParameter.tag file (see page 104). You use this template to change the value that is passed to the Visual Warehouse program when this particular business view runs.

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

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

Tokens

Table 43 provides information about each token in the template.

Table 43. VWPProgramInstanceParameter.tag tokens

Token	Description	Allowed values	Window or notebook: field
	Entity parameters		
**VWPProgramInstanceParameterName*	The name or description of a parameter that is to be passed to a Visual Warehouse program. This token is required.	A text string, up to 80 bytes in length.	Program: Parameter Name
**VWPProgramInstanceParameterOrder*	A number, starting with 0, that indicates the order of the parameter in the parameter list. This token is required.	A numeric value.	Program: Parameter List
**VWPProgramInstanceParameterData*	Data that is passed to the Visual Warehouse program as the value of the parameter. This token is required.	A text string or a numeric value up to 240 bytes in length.	Program: Parameter Text

Table 43. VWPProgramInstanceParameter.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**VWPProgramInstanceParameterKey*	A key that uniquely identifies this program parameter instance. The key must be unique from all other keys in the interchange file. **Tip:** Finish processing the VWPProgramInstanceParameter.tag template before increasing the value of the key. This token is required.	A numeric value, up to 80 bytes in length.	None
	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 numeric value, up to 80 bytes in length.	None
**CurrentCheckPointID++*	An 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

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

Table 44. Example values for VWPProgramInstanceParameter.tag tokens

Token	Example value
**VWPProgramInstanceParameterName*	DB2 UDB user ID
**VWPProgramInstanceKey*	070000
**VWPProgramInstanceParameterOrder++*	1
**VWPProgramInstanceParameterData*	my_userid
**VWPProgramInstanceParameterKey*	012994
**VWPProgramInstanceKey*	070001
**CurrentCheckPointID++*	12

VWPProgramTemplate.tag

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

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

Tokens

Table 45 provides information about each token in the template.

Table 45. VWPProgramTemplate.tag tokens

Token	Description	Allowed values	Window or notebook: field
	Entity parameters		
**VWPProgramTemplateName*	The name of the Visual Warehouse program template. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Program: Business Name

Table 45. VWPProgramTemplate.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**VWPTemplateDescription*	The short description of the Visual Warehouse program and what it does. This token is optional.	A text string, up to 200 bytes in length.	Program: Description
**VWPTemplateNotes*	The long description of the Visual Warehouse program and what it does. This token is optional.	A text string, up to 32700 bytes in length.	Program: Notes
**VWPTemplateExecutableName*	Fully qualified program name of the Visual Warehouse program that is to run when the business view runs. If the Visual Warehouse program is installed in the system path, the Visual Warehouse program name need not be fully qualified. This token is required.	A file name, up to 240 bytes in length.	Program: Fully Qualified Program Name

Table 45. VWPProgramTemplate.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**VWPProgramTemplateType*	The type of program. This token is required.	One of the following values: **ISV_PROGRAMTYPEDLL** The Visual Warehouse program is loaded from a dynamic load library (DLL) or is a load module. **ISV_PROGRAMTYPECOMMAND** The Visual Warehouse program is a command file. **ISV_PROGRAMTYPEEXECUTABLE** The Visual Warehouse program is an executable file.	Program: Program Executable Type
**VWPProgramTemplateFunctionName*	If the value of **VWPProgramTemplateType* is ISV_PROGRAMTYPEDLL, the name of the entry point in the DLL that Visual Warehouse is to invoke. This token is required if the value of **VWPProgramTemplateType* is ISV_PROGRAMTYPEDLL.	A text string, up to 80 bytes in length.	Program: Function Name
	Relationship parameters		
**VWPGroup*	The name of the group that is to contain the Visual Warehouse program. This token is required.	A text string, up to 80 bytes in length.	Program: Program Group

Table 45. VWPProgramTemplate.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**CurrentCheckPointID++*	An index, starting with 0, that increases each time it is substituted in a token. This token is required.	A numeric value.	None
**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.	Warehouse: Selected Agent Sites

Examples of values

Table 46 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 46. Example values for VWPProgramTemplate.tag tokens

Token	Example value
**VWPProgramTemplateName*	My ISV program
**VWPTemplateDescription*	This program exports data from an ODBC database.
**VWPTemplateNotes*	This program will....
**VWPTemplateExecutableName*	c:\ISV\BIN\MYPROG.EXE
**VWPTemplateType*	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 Visual Warehouse is to pass to a Visual Warehouse program.

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

Use this template with the VWPProgramTemplate.tag file ("VWPProgramTemplate.tag" on page 101). This template defines the relationship between the parameter and its Visual Warehouse program definition (VWPProgramTemplate.tag).

Tokens

Table 47 provides information about each token in the template.

Table 47. VWProgramTemplateParameter.tag tokens

Token	Description	Allowed values	Window or notebook: field
	Entity parameters		
**VWPTemplateParameterName*	The name or description of a parameter that is to be passed to a Visual Warehouse program. The name must be unique within the Visual Warehouse program. This token is required.	A text string, up to 80 bytes in length.	Program: Parameter Name
**VWPTemplateParameterOrder*	A number, starting with 0, that indicates the order of the parameter in the parameter list. This token is required.	A numeric value.	Program: Parameter List
**VWPTemplateParameterData*	Data that is passed to the Visual Warehouse program as the value of the parameter. This token is required.	A text string or a numeric value up to 240 bytes in length.	Program: Parameter Text

Table 47. VWProgramTemplateParameter.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**VWPTemplateParameterKey*	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.	None
	Relationship parameters		
**VWPTemplateName*	The name of the Visual Warehouse program that is to use this parameter. This token is required.	A text string, up to 80 bytes in length.	Program: Business Name
**CurrentCheckPointID++*	An 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 48 provides example values for each token to illustrate the kind of metadata you might provide for each token.

Table 48. Example values for VWPProgramTemplateParameter.tag tokens

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

WarehouseDataBase.tag

Use this template to define target warehouse databases to import into Visual Warehouse.

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 49 provides information about each token in the template.

Table 49. WarehouseDataBase.tag tokens

Token	Description	Allowed values	Window or notebook: field
		Entity parameters	
**DatabaseName*	The name of the database. The name must be unique within the Visual Warehouse control database. This token is required.	A text string, up to 80 bytes in length.	Warehouse: Target Warehouse Name
**DatabaseDescription*	The short description of the database. This token is optional.	A text string, up to 200 bytes in length.	Warehouse: Description
**DatabaseNotes*	The long description of the database. This token is optional.	A text string, up to 32700 bytes in length.	Warehouse: Notes

Table 49. WarehouseDataBase.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**DatabaseContact*	The person to contact for information about this database. This token is optional.	A text string, up to 64 bytes in length.	Warehouse: Contact
**DatabaseServerName*	The name of the server on which the database resides. This token is optional.	A text string, up to 64 bytes in length.	None
**DatabasePhysicalName*	The physical database name of the database as defined to the DBMS. This token is required.	A text string, up to 40 bytes in length.	Warehouse: Database

Table 49. WarehouseDataBase.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
**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	Resource Type
**DatabaseTypeExtended*	The type of AS/400 machine or file. This token is required.	One of the following values: **ISV_IR_DB2400CISC** DB2 for 400 for CISC machines **ISV_IR_DB2400RISC** DB2 for 400 for RISC machines **ISV_IR_FFLanLocalCmd** Local flat file **ISV_IR_FFLanFTPCopy** Local flat file sent using FTP from a remote system	Warehouse: Type
**DatabaseUserid*	The user ID with which to access the database. This token is optional.	A text string, up to 36 bytes in length.	Warehouse: User ID

Table 49. WarehouseDataBase.tag tokens (continued)

Token	Description	Allowed values	Window or notebook: field
		Relationship parameters	
**SecurityGroup*	The security group in which to create 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. Specify `ISV_DEFAULTSECURITYGROUP` for the default security group.	Warehouse: Selected Security Groups
**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.	Warehouse: Selected Agent Sites
**CurrentCheckPointID++*	An 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 50 provides examples of values for each token to illustrate the kind of metadata you might provide for each token.

Table 50. 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
**DatabasePhysicalName*	FINANCE

Table 50. example values for WarehouseDataBase.tag tokens (continued)

Token	**Example value**
**DatabaseType*	DB2 Family
**DatabaseTypeExtended*	ISV_DEFAULTVALUE
**DatabaseUserid*	DB2ADMIN
**SecurityGroup*	ISV_DEFAULTSECURITYGROUP
**AgentSite*	My agent site
**CurrentCheckPointID++*	6

WarehouseDataBase.tag

Chapter 8. Visual Warehouse metadata

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

Table 51 describes the mapping between each object in the tag language file and the corresponding logical object in Visual Warehouse.

Table 51. Logical objects for source and target databases

Object in tag language file	Visual Warehouse logical object	Description	See:
DATABASE	An information resource or warehouse	A data source or data target	"DATABASE object"
TABLE	A table, file, or segment	A table in a source or target database, or a file	"TABLES object" on page 120
COLUMN	A column or field	A column in a table or a field in a file	"COLUMN object" on page 124

Visual Warehouse 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 52 on page 114 provides information about the properties of the DATABASE object.

Table 52. Properties of the DATABASE object

Tag language property name	Description	Allowed values	Window or notebook: field
NAME	The business name of the resource.	A text string, up to 80 bytes in length.	Information Resource: Name Warehouse: Name
DBNAME	The physical database name as defined to the DBMS. This value is null for generic ODBC databases, Sybase databases, generic ODBC databases, and file systems.	A text string, up to 40 bytes in length.	Information Resource: Database Information Resource: Data Source Name Warehouse: Name
SHRTDESC	The short description of the resource.	A text string, up to 200 bytes in length.	Information Resource: Description Warehouse: Description
LONGDESC	The long description of the resource.	A text string, up to 32700 bytes in length.	Information Resource: Notes Warehouse: Notes
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	Information Resource: Type Warehouse: Type

Table 52. Properties of the DATABASE object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
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** Microsof®t SQLServer **50** Informix **60** User Defined ODBC	Information Resource: Type Warehouse: Type
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	Information Resource: Type Warehouse: Type

Table 52. Properties of the DATABASE object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
ISWH	Flag indicating whether this resource is a warehouse.	One of the following values: **Y** This resource is a warehouse. **N** This resource is an information resource.	None
LOCATION	Descriptive information about the location of the resource, such as the room number.	A text string, up to 64 bytes in length.	Information Resource: Location Warehouse: Location
USERID	The user ID that Visual Warehouse uses to connect to the resource.	A text string, up to 36 bytes in length.	Information Resource: User ID Warehouse: User ID
CONTACT	The name of the person who is responsible for the resource.	A text string, up to 64 bytes in length.	Information Resource: Contact Warehouse: Contact
USEODBC	Flag indicating whether to use the user-supplied connect string or have Visual Warehouse generate the string. Use N for files.	One of the following values: **Y** Use the user-defined connect string. **N** Have Visual Warehouse generate the connect string.	Information Resource: Customize ODBC ConnectString
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.	Information Resource: ODBC Connect String
CODEPAGE	The code page used to create the database. This property is for compatibility with previous releases of Visual Warehouse.	A numeric value.	None

Table 52. Properties of the DATABASE object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
SERVER	One of the following types of values: • The name of the server workstation that contains a database. This identifier indicates whether the databases with the same name are the same physical database. • The host name for the Flat File LAN FTP type. • Otherwise, null.	A text string, up to 64 bytes in length.	Information Resource (database): System Name Information Resource (file): Host Name Warehouse: System Name
PREACCMD	If the resource is a local Flat File LAN resource, a command to run to access the remote file.	A text string, up to 64 bytes in length.	Information Resource: Local File
POSTACMD	If the resource is a local Flat File LAN resource, a command to run after access of the remote file.	A text string, up to 64 bytes in length.	Information Resource: Local File
RETRYCNT	The number of times Visual Warehouse will try to extract data from this resource in case of an error.	A numeric value.	Information Resource: Default Retry Count Warehouse: Default Retry Count
RETRYINT	The time that is to elapse between attempts to extract data.	A numeric value.	Information Resource: Default Retry Interval Warehouse: Default Retry Interval

Figure 15 on page 118 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 15. Target DATABASE object instance

Figure 16 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 16. Source file DATABASE object instance

Relationships

Table 53 lists 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 53, the values 1 and M indicate that one database can relate to many tables, but a table can only relate to one database.

Table 53. 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 contained in the database or file system.

Figure 17 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 17. Linking DATABASE object instance to TABLES object instance

TABLES object

This object contains metadata about a Visual Warehouse source table, segment, or file, or a target table. It is associated with a DATABASE object (see “DATABASE object” on page 113).

Properties

Table 54 provides information about the properties of the TABLES object.

Table 54. Properties of the TABLES object

Tag language property name	Description	Allowed values	Window or notebook: field
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: Name
SHRTDESC	The short description of the file or segment.	A text string, up to 200 bytes in length.	Table: Description
LONGDESC	The long description of the table.	A text string, up to 32700 bytes in length.	Table: Notes

Table 54. Properties of the TABLES object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
DBNAME	The business name of the resource that contains this table or file.	A text string, up to 80 bytes in length.	Information Resource: Name Warehouse: Name
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.	Table: Notes
TABLES	The physical table, file, or segment name as defined to the DBMS or file system. For files and IMS segments, this value is same as the value of NAME.	A text string, up to 80 bytes in length.	Table: Name
TBLISBIN	Flag indicating 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.	File: File Transfer Mode
TBLNAMESP	The name of the DB2 table space.	A text string, up to 90 bytes in length.	None
TBLFTYPE	For files, the type of the file.	One of the following values: **1** Fixed **2** Comma **3** Tab **4** Character	File: File Type
TBLL1NAM	Flag indicating 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.	File: First Row Contains Column Names

Table 54. Properties of the TABLES object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
CHARDELM	For files, the character separator if the file type is Character.	A text string that is 1 byte in length.	File: File Delimiter Character
CREATYPE	The method used to define the table in Visual Warehouse.	One of the following values: **1** The table was defined manually. **2** The table definition was imported from the DBMS. **3** The table definition was imported from DataGuide. **4** The table was created by Visual Warehouse for a business view when the business view was promoted to test status.	Information Resource: Tables Warehouse: Tables

Figure 18 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 18. TABLES object instance for a relational table

Figure 19 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 19. TABLES object instance for a file

Relationships

Table 55 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 55. 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 20 on page 124 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 20. Linking TABLES object instance to DATABASE object instance

Figure 21 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 21. Linking TABLES object instance to COLUMN object instance

COLUMN object

This 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 120).

Properties

Table 56 provides information about the properties of the COLUMN object.

Table 56. Properties of the COLUMN object

Tag language property name	Description	Allowed values	Window or notebook: field
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.	Table: (Column) Name
SHRTDESC	The short description of the column or field.	A text string, up to 200 bytes in length.	Table (Column): Description
LONGDESC	The long description of the column or field.	A text string, up to 32700 bytes in length.	Table: (Column) Notes

Table 56. Properties of the COLUMN object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
DATATYPE	The ODBC data type to which the DBMS data type maps. Visual Warehouse derives the data type from the native data type.	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	Table: Column Information
LENGTH	The length of the column or field.	A numeric value.	Table: Length/Precision
SCALE	The precision of the column or field for columns or fields with a decimal data type.	A numeric value.	Table: Length/Precision
POSNO	An index, starting with 0, of the column or field in the row of the table or file.	A numeric value.	Table: Column Information
NULLS	Flag indicating 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.	Table: Column Information

Table 56. Properties of the COLUMN object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
ISTEXT	Flag indicating 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.	Table: Column Information
DBNAME	The business name of the resource that contains this table or file.	A text string, up to 80 bytes in length.	Information Resource: Name Warehouse: Name
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.	Table: Notes
TABLES	The physical table, file, or segment name as defined to the DBMS or file system. For files and IMS segments, this value is same as the value of NAME.	A text string, up to 80 bytes in length.	Table: Name
NATIVEDT	Native data type of the column or field.	The data type for the column as defined to the DBMS. 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 DBMS datatypes to ODBC data types, see the Visual Warehouse online help.	Table: Native Type

Table 56. Properties of the COLUMN object (continued)

Tag language property name	Description	Allowed values	Window or notebook: field
TRANSNAM	Transformer property that produces or uses this column or field. For example, if the column is used as a grouping column, the value of this property is Grouping.	A text string, up to 80 bytes in length.	None

Figure 22 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 22. COLUMN object instance

Relationships

Table 57 lists the relationship in which the COLUMN 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.

Table 57. 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	Table with which this column is associated.

COLUMN object

Figure 23 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 23. Linking COLUMN object instance to TABLES object instance

Chapter 9. DataGuide system tables and metadata models

The following tables are defined for DataGuide 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.

Column name	Data type	Description	Nullable	NLS
RELTYPE	CHAR(1)	Relation type: **A** Attachment relation **L** Link relation **M** Comments relation	N	S
SOURCE	CHAR(16)	The FLGID that represents the source object instance.	N	S

Column name	Data type	Description	Nullable	NLS
TARGET	CHAR(16)	The FLGID that represents the target object instance	N	S
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

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 might 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.

Column name	Data type	Description	Nullable	NLS
TAGFNAME	VARCHAR(240)	The name of the tag language file (without the path information).	N	B
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.	N	B
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.	N	N
USERID	CHAR(8)	The user ID of the DataGuide administrator.	N	B
ENTSAVED	INTEGER	The total number of entries that have been saved in the save area.	N	N

Column name	Data type	Description	Nullable	NLS
SAVEAREA	LONG VARCHAR	Storage area for a list of object type names. Each object type name is 8 bytes.	N	S
Note: Nullable denotes a column may have a null character as value. Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

FLG.COMMENTS table

The FLG.COMMENTS table contains all the comments on objects in the DataGuide information catalog.

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

INSTIDNT is the primary key of the table.

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

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

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	This six-digit object type ID, generated by DataGuide, represents a specific object type in DataGuide.	N	S
INSTIDNT	CHAR(10)	The unique instance ID generated by DataGuide. It is the second part of the FLGID, the 10-digits serial number that will uniquely identify this instance within its own object type.	N	S
NAME	VARCHAR(80)	The name entered by the DataGuide user to identify each user defined object instance.	N	B
UPDATIME	CHAR(26)	The date and time of the metadata creation or last update. This date is generated by DataGuide.	Y	N
UPDATEBY	CHAR(8)	The user ID of the DataGuide administrator who last updated the instance.	Y	B

Column name	Data type	Description	Nullable	NLS
CREATOR	CHAR(8)	The creator of the Comments object. The system will set the creator to the current user ID.	N	B
CREATSTP	CHAR(26)	A timestamp indicating the date and time the Comments object instance was created. This time stamp is supplied by the system when the instance is created.	N	N
STATUS	CHAR(80)	The status of the comment. The user can design their own conventions for this value.	Y	B
ACTIONS	VARCHAR(250)	Specifies what action the user should take.	Y	B
EXTRA	VARCHAR(80)	This column is used for extra information.	Y	B
Note: Nullable denotes a column may have a null character as value. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: None				

FLG.EXCHANGE table

The FLG.EXCHANGE table is used to keep track of the object sychronized between DataGuide, Visual Warehouse, 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.

Column name	Data type	Description	Nullable	NLS
PRODUCT	VARCHAR(40)	The combination of product, version, and release numbers.	N	S
OBJNAME	VARCHAR(200)	The object name, for example, business view.	N	B
IMPDATE	TIMESTAMP	The import timestamp.	N	N

Column name	Data type	Description	Nullable	NLS
OBJTYPE	CHAR(5)	OBJTYPE can be one of the following values: • IR represents source metadata exchanged • DR represents target metadata • BV represents business view metadata • OLAP represents OLAP metadata	N	S
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

FLG.HISTORY table

The FLG.HISTORY table is used to keep track of object instances that have been deleted from DataGuide and Visual Warehouse.

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

The HISSEQ column is the primary key of the table.

Column name	Data type	Description	Nullable	NLS
HISSEQ	TIMESTAMP	The sequence number of the delete history.	N	N
HISTYPE	INTEGER	The type of the delete history. • A value of 1 in this column indicates a deletion from DataGuide. • A value of 2 in this column indicates a deletion from Visual Warehouse.	N	N
HISTAG	LONG VARCHAR	This column will store the identifier of the object to be deleted.	Y	B
Note: Nullable denotes a column may have a null character as value. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: None				

FLG.NAMEINST table

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

The FLGID column is the primary key of the table.

The INSTNAME and TYPENAME columns are indexes of the table.

Column name	Data type	Description	Nullable	NLS
FLGID	CHAR(16)	The 16-character object instance ID.	N	S
TYPENAME	VARCHAR(80)	The external name of the object type.	N	B
INSTNAME	VARCHAR(80)	The external name of an object instance.	N	B
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

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 DataGuide.

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

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

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

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	The six-digit object type ID generated by DataGuide. The ID represents a specific object type in DataGuide.	N	S

Column name	Data type	Description	Nullable	NLS
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.	N	S
DPNAME	CHAR(8)	The unique object type name within an information catalog.	N	S
NAME	VARCHAR(80)	The external name of this object type.	N	B
CATEGORY	CHAR(1)	The DataGuide categories: Elemental E, Grouping G, Program P, Contact C, Dictionary D, Support S, and Attachment A.	N	S
CREATOR	CHAR(8)	The user ID of the DataGuide 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.	Y	B
UPDATIME	CHAR(26)	The date and time of the object type that was created or that had its properties extended.	Y	S
UPDATEBY	CHAR(8)	The user ID of the DataGuide administrator who last extended the object type (appended properties).	Y	B
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 DataGuide user. It is accessed and updated by the Create Instance IPI only.	N	N
OBJICON	LONG VARCHAR FOR BIT DATA	The icon bitmap corresponding to the object type.	N	N

Note:

Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable.

NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable

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.

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	The six-digit, object type ID generated by DataGuide, represents a specific object type in DataGuide.	N	S
INSTIDNT	CHAR(10)	The unique instance ID generated by DataGuide. 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.	N	S
PHYPRPNM	CHAR(8)	The original property or column name defined by the user.	N	S
SEQNO	SMALLINT	A sequence number to keep track of how many rows reflect the same incoming source.	N	N
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.	N	B

Note:

Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable.

NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable

FLG.PARMS table

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

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

Column name	Data type	Description	Nullable	NLS
VERSION	CHAR(20)	Version of the DataGuide information catalog, for example, V1R0M0 or V1R1M0; which is populated at the installation or migration time.	Y	S
LOGONMSG	VARCHAR(254)	DataGuide logon message, for example, ″Welcome to DataGuide!″	Y	B
CODEPAGE	CHAR(4)	Code page number of the information catalog.	Y	S
LANGUAGE	CHAR(4)	Language code, for example, ENU (US English). It is loaded from a string file.	Y	S
DTOKEN	CHAR(1)	The default token of the DataGuide 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.	Y	S
ENVSIZE	SMALL INTEGER	Database server environment size. This value is set during installation, and is used to specify the proper name length for DataGuide tables, columns, and indexes. This value can be 10 for OS/400 DBMS, 18 for most other IBM relational databases, and up to a maximum of 30 bytes for non-IBM databases.	Y	N
LASTYPID	INTEGER	The last system-generated ID for an object type. The ID is accessed and updated by the Create Registration IPI only.	Y	N
LISTMAX	INTEGER	The maximum number of retrievable objects from a listing or search result.	Y	N
ISTGROUP	CHAR(8)	The index storage group name for the DB2 for OS/390® database.	Y	S
TSTGROUP	CHAR(8)	The table storage group name for the DB2 for OS/390 database.	Y	S
MDBNAME	CHAR(8)	The DB2 for OS/390 database name.	Y	S
TBSPAC32	CHAR(8)	The 32 KB table space name for the DB2 for OS/390 database.	Y	S
TBSPAC04	CHAR(8)	The 4 KB table space name for DB2 for OS/390 database.	Y	S

Column name	Data type	Description	Nullable	NLS
PARMFLAG	INTEGER	A flag indicator. **FLG_PARMS_RECORD_DELETE_HISTORY** Records the delete history. **FLG_PARMS_MVS_FOLD_UP** Saves the object values in upper case in the DB2 for OS/390 information catalog. You can search these values in uppercase or lowercase in DataGuide.	Y	N
CMTSTAT	VARCHAR(800)	This column stores a list of comments status. Each status is 80 bytes.	Y	B
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

FLG.PROGRAMS table

The FLG.PROGRAMS table is used to keep track of all program objects in DataGuide.

INSTIDNT 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.

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

Column name	Data type	Description	Origin	NLS
OBJTYPID	CHAR(6)	The six-digit object type ID, generated by DataGuide, represents a specific object type.	N	S
INSTIDNT	CHAR(10)	The unique instance ID generated by DataGuide. It is the second part of the FLGID, the 10-digit serial number that uniquely identifies this instance within its own object type.	N	S

Column name	Data type	Description	Origin	NLS
NAME	VARCHAR(80)	This name is entered by the DataGuide user to identify each user-defined object instance.	N	B
UPDATIME	CHAR(26)	The date and time of metadata creation or last update. This is generated by DataGuide.	Y	S
UPDATEBY	CHAR(8)	The user ID of the DataGuide administrator that last updated the instance.	Y	B
UUICLASS	CHAR(25)	The part1 name of the universal unique identifier (UUI)	N	B
UUIQUAL1	VARCHAR(48)	The part2 name of the (UUI).	N	B
UUIQUAL2	VARCHAR(48)	The part3 name of the (UUI)	N	B
UUIQUAL3	VARCHAR(48)	The part4 name of the (UUI).	N	B
UUIDENT	VARCHAR(70)	The part5 name of the (UUI).	N	B
HANDLES	CHAR(8)	The Object type that this program handles.	Y	S
STARTCMD	VARCHAR(250)	The program name to be invoked. The program can have the extension of .exe, .cmd, .com, or .bat.	N	B
PARMLIST	VARCHAR(1800)	If a parameter list is required to handle object instances, the value of the parameter is specified by the HANDLES property.	Y	B
SHRTDESC	VARCHAR(250)	The short description of the program.	Y	B
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

FLG.PROPERTY table

The FLG.PROPERTY 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 DataGuide object types and object type properties, see “Chapter 10. DataGuide object types” on page 155.

OBJTYPID is the index of the table.

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	System-generated ID that is a unique 6 digits for each object type.	N	S
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.	N	S
PROPNAME	VARCHAR(80)	The external name of this object type property.	N	B
DATATYPE	CHAR(30)	Property data type, CHAR, VARCHAR, LONG VARCHAR and TIMESTAMP.	N	S
LENGTH	INTEGER	Property length.	N	N
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 (DataGuide) generated value	N	S
UUISEQNO	CHAR(1)	The UUI sequence number of the property in the object type.	Y	S
PROPSEQ	INTEGER	The sequence number of the property	N	N

Note:
Nullable = N: Not Nullable Y: Nullable
Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable.
NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

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.

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

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

Column name	Data type	Description	Nullable	NLS
RELTYPE	CHAR(1)	Relation type: **C** Contains **T** Contact	N	S
SRCCAT	CHAR(1)	Category of the source object.	N	S
SOURCE	CHAR(16)	SOURCE is the FLGID that represents the source object instance.	N	S
SRCTNAME	VARCHAR(80)	SRCTNAME is the external name of the source object type.	N	B
SRCINAME	VARCHAR(80)	SRCINAME is the external name of the source object instance.	N	B
TRGCAT	CHAR(1)	Category of the target object.	N	S
TARGET	CHAR(16)	TARGET is the FLGID that represents the target object instance	N	S
TRGTNAME	VARCHAR(80)	TRGTNAME is the external name of the target object type.	N	B
TRGINAME	VARCHAR(80)	TRGINAME is the external name of the target object instance.	N	B

Note:

Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable.

NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable

FLG.USERS table

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

USERTYPE and DGUSER form the primary key of the table.

DGUSER is an index of the table.

Column name	Data type	Description	Nullable	NLS
DGUSER	CHAR(8)	The user ID of the DataGuide administrator. The ID is entered at installation.	N	B

Column name	Data type	Description	Nullable	NLS
USERTYPE	CHAR(1)	Type of DGUSER. The type can be a DataGuide Administrator, a user with special update privileges, or user. This value is set during installation.	N	S
ACTIVEKA	CHAR(1)	A flag to indicate the DataGuide administrator who is currently logged on to DataGuide. Only one DataGuide administrator can be logged on at a time.	Y	S
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

FLG.WINICON table

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

OBJTYPID is the primary key of the table.

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	The six-character object type ID.	N	S
OBJICON	LONG VARCHAR FOR BIT DATA (30000)	The bitmap for the Windows icon.	Y	N
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

DataGuide metadata models

The following sections describe the DataGuide metadata models. “Model for DataGuide system tables” on page 143 describes the relationships between DataGuide system tables. “Logical metadata model” on page 147 describes the relationships between objects in DataGuide object type categories.

Model for DataGuide system tables

The following illustrations show the relationships between the different DataGuide system tables as well as the object type tables. For example, a relationship can be a join between two columns. The following DataGuide 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.

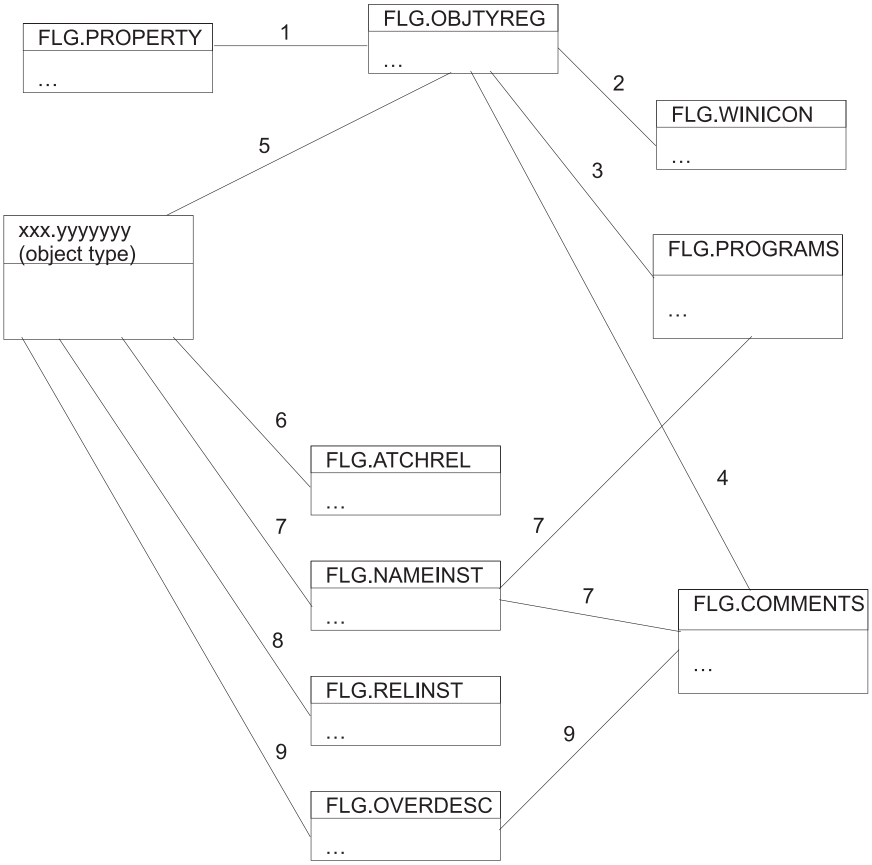
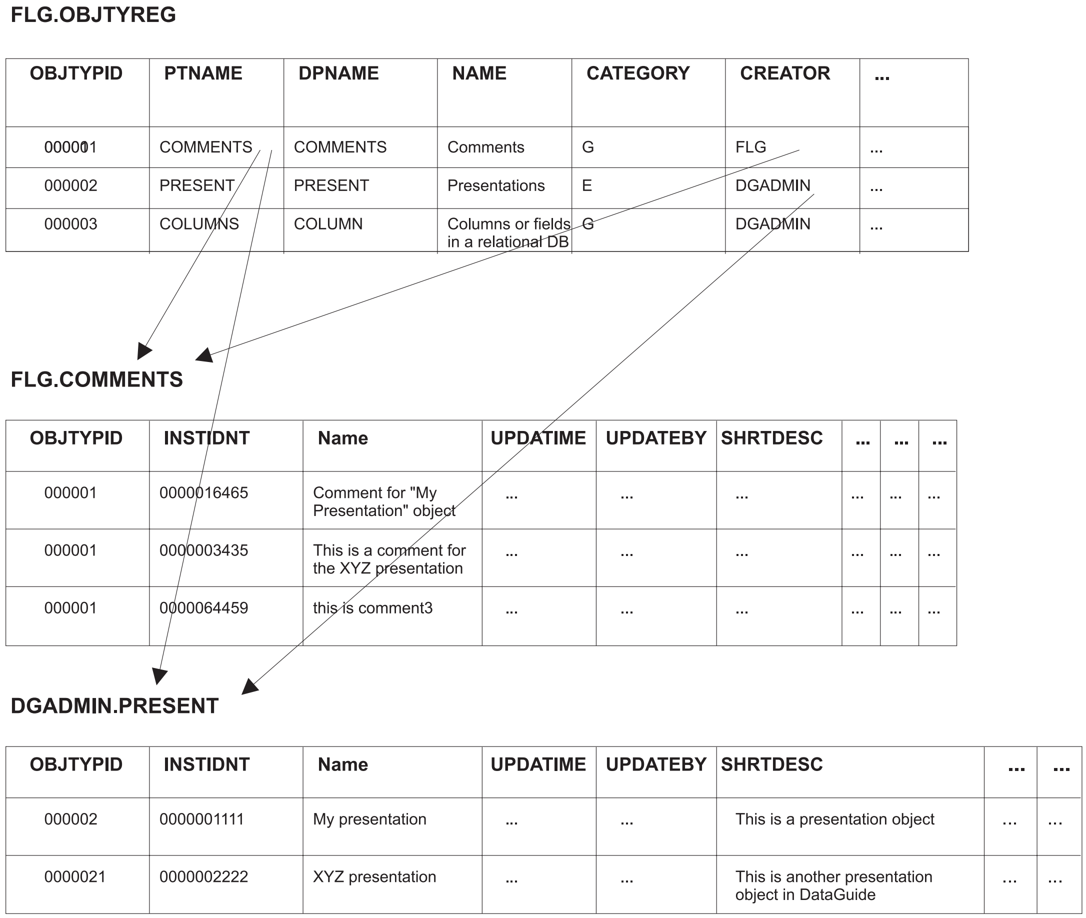


Figure 24. DataGuide system tables

Notes to Figure 24

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 25 on page 145, 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 DataGuide	...	...

Figure 25. Relationship between table FLG.OBJTYREG and the object type table

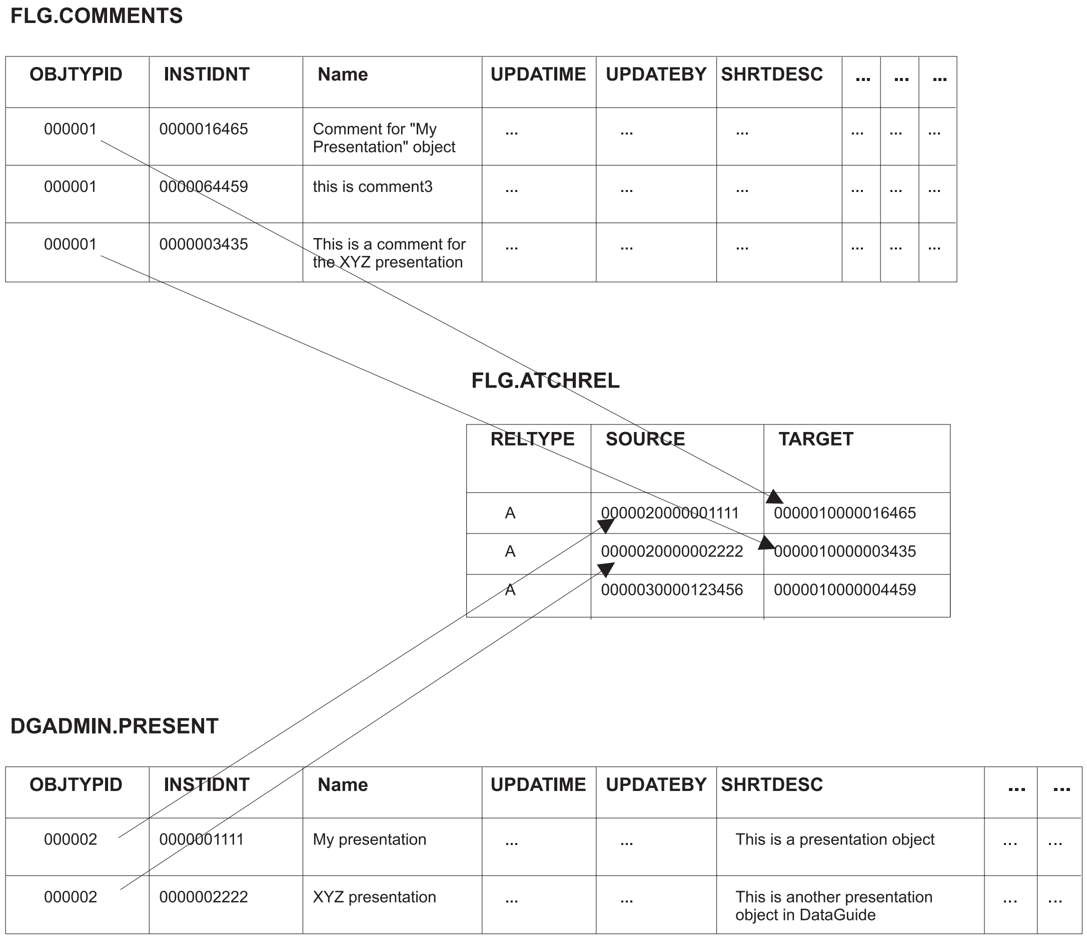
5. The relationship between the FLG.OBJTYPREG table and an object type table is derived by concatenating the PTNAME and CREATOR columns of the FLG.OBJTYPREG table. The resulting name is the name of the object type table.

   For example in Figure 25, 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 26 on page 146, the object type and instance ID for DGADMIN.PRESENT is 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 DataGuide	...	...

Figure 26. 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” on page 147 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 DataGuide object types” on page 159 for more information on DataGuide 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 a DataGuide category. An object type's *category* affects how DataGuide handles it. Except for the Program and Attachment categories, you can create object types in any of the following DataGuide categories:

Grouping
Object types that can contain other object types.

Elemental
Non-Grouping object types that are the building blocks for other DataGuide 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 DataGuide 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 DataGuide 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 58 on page 148 summarizes the relationships among DataGuide's object type categories. Figure 27 on page 149 shows a graphical representation of the relationships.

Table 58. DataGuide 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 DataGuide in the sample information catalog (see “Predefined DataGuide object types” on page 159 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 DataGuide to fit your organization’s needs (see “Updating an object type using DataGuide tag language” on page 51 for information about modifying an object type).
- Create your own object types.

Figure 27 on page 149 shows how objects within DataGuide 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 10. DataGuide object types” on page 155 for more information on extendible object types.

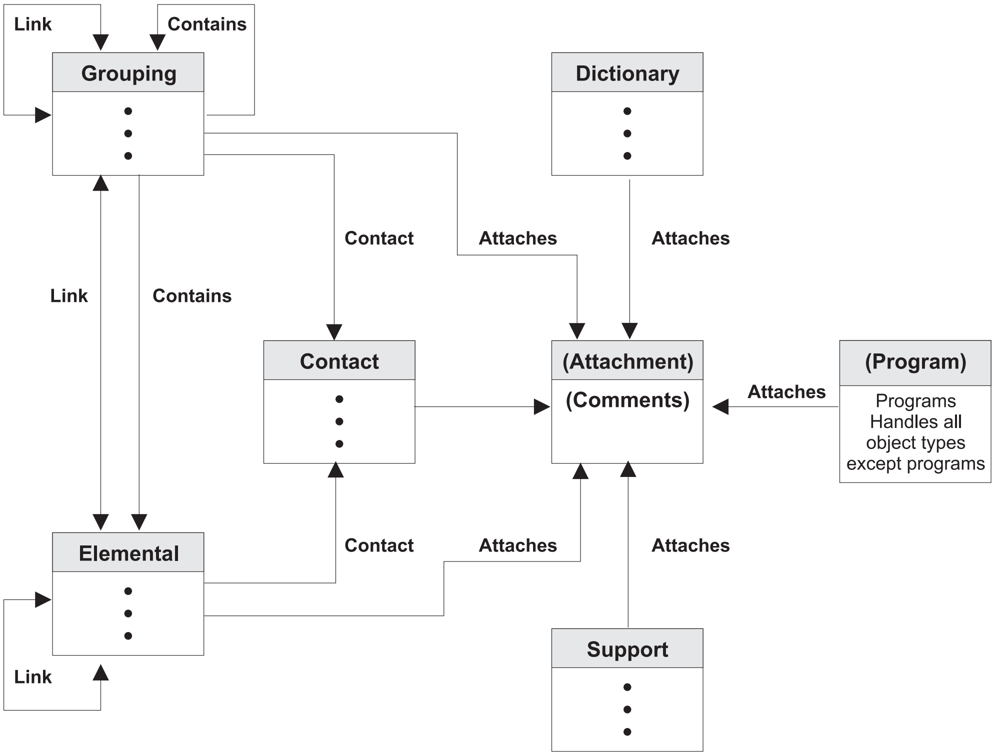


Figure 27. Relationships between object type categories

In Figure 27, 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 150) and the properties that you want (from step 2 on page 150), 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 151. 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 154.

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 10. DataGuide object types

This chapter describes detailed information about DataGuide object types.

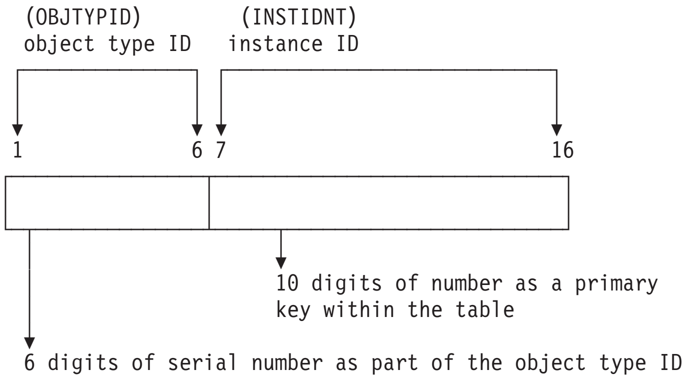
Default properties for all object types

DataGuide 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 DataGuide; some are required; and some are optional.

FLGID

An ID, generated by DataGuide, 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

Name Name of the business view. 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 DataGuide 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 DataGuide administrator or DataGuide user with special privileges who last updated the instance. For Attachment objects, this field can be the user ID of a DataGuide user.

Default property summary

The DataGuide administrator can use the predefined template to create an object type. The DataGuide administrator can append attributes to the template to customize it for the organization. The predefined template has several optional fields. The following table shows the default properties.

Column name	Data type	Description	Nullable	NLS
OBJTYPID	CHAR(6)	This six-digit object type ID, generated by DataGuide, represents a specific object type.	N	S
INSTIDNT	CHAR(10)	The unique instance ID generated by DataGuide. It is the second part of the FLGID, the 10-digit serial number that uniquely identifies this instance within its own object type.	N	S
NAME	VARCHAR(80)	This name is entered by the DataGuide user to identify each user-defined object instance in the product.	N	B
UPDATIME	CHAR (26)	The date and time of metadata creation or last update. This is generated by DataGuide.	N	S
UPDATEBY	CHAR(8)	The user ID of the DataGuide administrator or the DataGuide user with special update privileges who last updated the instance. For attachment objects this field might be the user ID of the DataGuide user. This is generated by DataGuide.	N	B
Note: Nullable denotes a column may have a null character as value. A ″Y″ indicates a column is nullable. An ″N″ indicates the column is not nullable. NLS = S: SBCS D: DBCS B: Both SBCS and DBCS N: Not applicable				

Rules to support 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 DataGuide 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 DataGuide 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 to create, delete, and update is identical for all object types, except for the PROGRAMS and COMMENTS object types.

The PROGRAMS object type is predefined by DataGuide 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 DataGuide 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 DataGuide.
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 DataGuide information catalog during the import process.
5. If the property has a data type such as LONG VARCHAR, DataGuide 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. DataGuide 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, so it’s a good idea to keep the length of the VARCHAR to a reasonable size.

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

   DataGuide 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, DataGuide 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 rules

1. DataGuide 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 is 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 DataGuide defined categories are described in “Logical metadata model” on page 147.

Objects are not required to have relationships. You can find all objects by using the DataGuide windows (see *Managing DataGuide*), the FLGSearch API or by viewing the FLG.NAMEINST table. See *DataGuide Programming Guide and Reference* for more information on DataGuide API’s. See “FLG.NAMEINST table” on page 134 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 140 for more information on the properties in the table.

Predefined DataGuide object types

DataGuide includes predefined object types that can be exchanged with metadata from other Visual Warehouse components and other MDIS-conforming products from IBM and other companies. This section describes all of the predefined DataGuide 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, visit the Meta Data Coalition’s Web site at http://www.MDCinfo.com.

DataGuide 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 DataGuide categories. This section describes how to create the sample information catalog. For details of DataGuide object type capabilities, see *Managing DataGuide*.

Table 59 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 59. Predefined data object types summary

Object type name	Description	Properties defined on page
Application data	Internal use only	171
Audio clips	Represents files that contain audio information	201
Business subject areas	Represents logical grouping of objects	173
Charts	Represents either printed or electronic charts	202
Columns or fields	Represents columns within a relational table, fields within a file, or fields within an IMS segment	174
Comments	Contains comments about other objects in the information catalog	216
Databases	Represents relational databases	176
DataGuide news	Conveys to end users information about changes to the information catalog	213
Dimensions within a multi-dimensional database	Represents dimensions within a multi-dimensional database	178
Documents	Represents books, manuals, and technical papers	203

Table 59. Predefined data object types summary (continued)

Object type name	Description	Properties defined on page
Elements	Represents MDIS Element objects that do not map directly to the “Columns or fields” object type	179
Files	Represents a file within a file system	181
Glossary entries	Represents definitions for terms used in the information catalog	211
Images or graphics	Represents graphic images, such as bitmaps	204
IMS database definitions (DBD)	Represents IMS database definitions	183
IMS program control blocks (PCB)	Represents IMS program control blocks	185
IMS program specification blocks (PSB)	Represents IMS program specification blocks	186
IMS segments	Represents IMS segments	187
Internet documents	Represents Web sites and other documents on the Internet that might be of interest	205
Lotus Approach queries	Represents available Lotus Approach queries for use with your organization’s data	205
Members within a multi-dimensional database	Represents a member within a multi-dimensional database	189
Multi-dimensional databases	Represents multi-dimensional databases	191
Online news services	Represents news and information services that can be accessed online	213
Online publications	Represents publications and other documents that can be accessed from online services	214
People to contact	Identifies a person or group that is responsible for single or multiple objects within the information catalog	210
Presentations	Represents printed or electronic presentations	206

Table 59. Predefined data object types summary (continued)

Object type name	Description	Properties defined on page
Programs that can be invoked from DataGuide objects	Defines an application capable of processing a particular object type	215
Records	Represents MDIS Record objects that do not map directly to the “Files” or “Relational tables or views” object type	193
Relational tables and views	Represents tables or views of relational databases	194
Subschemas	Represents logical groupings of records within a database	196
Transformations	Represents expressions or logic used to populate columns of data within the target relational database	198
Spreadsheets	Represents desktop spreadsheets (for example, Lotus 1-2-3 or Microsoft Excel spreadsheets)	207
Text-based reports	Represents either printed or electronic reports	208
Video clips	Represents files that contain video information	209

Predefined object type models

The DataGuide predefined object types follow the six data models shown in Figures 28 through 33.

Figure 28 shows the object types that participate in the relational model.

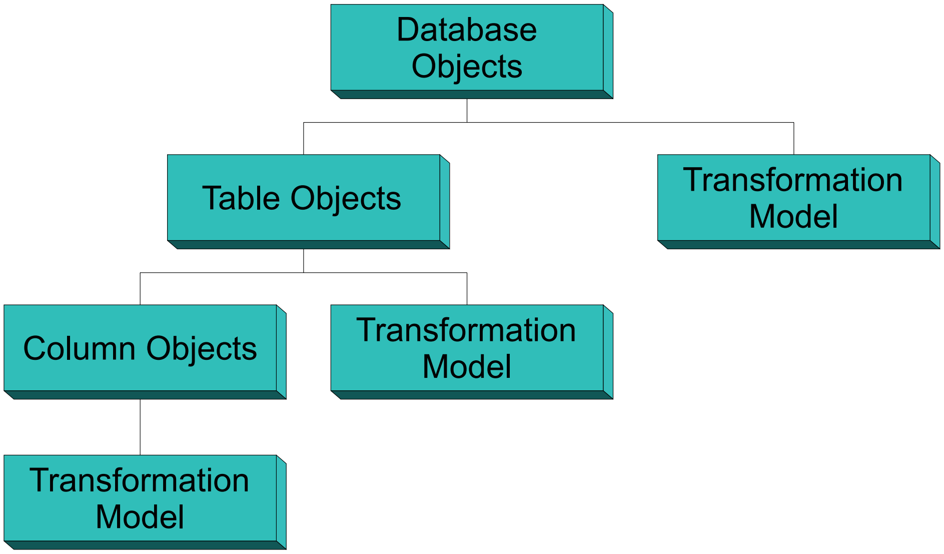


Figure 28. Relational model and the predefined object types

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

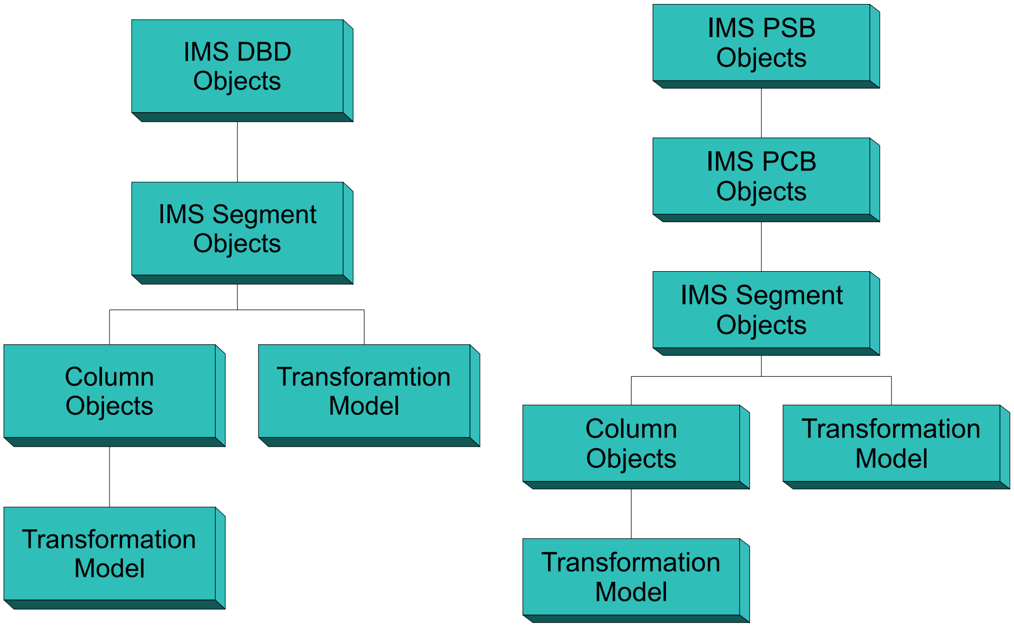


Figure 29. Hierarchical models and the predefined object types

Predefined object type models

Figure 30 shows the object types that participate in the file models.

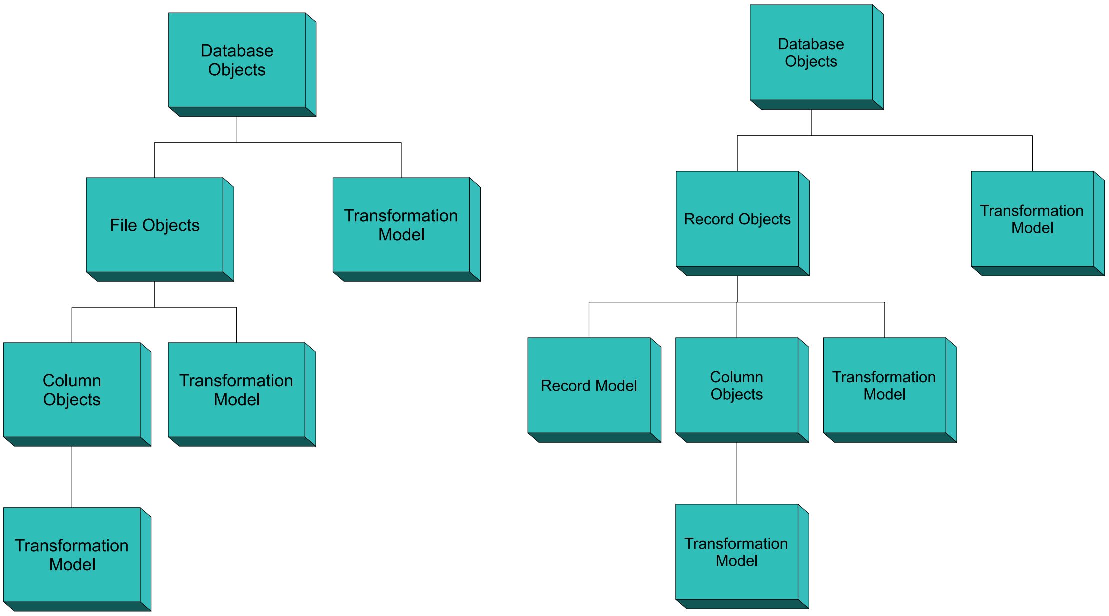


Figure 30. File models and the predefined object types

Figure 31 shows the object types that participate in the multi-dimensional (OLAP) model.

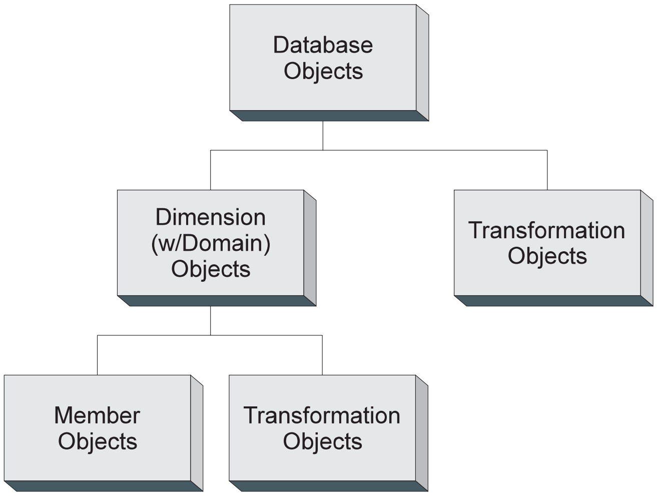


Figure 31. Multi–dimensional model and the predefined object types

Figure 32 shows the object types that participates in the transformation models.

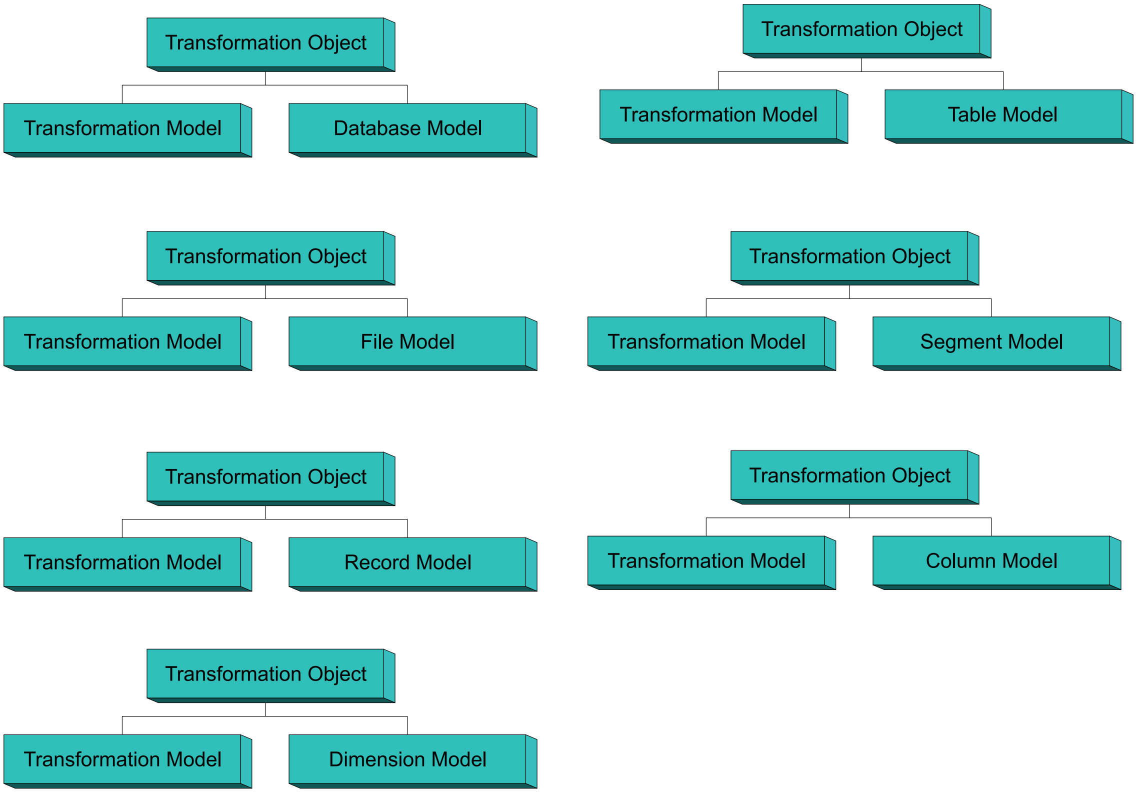


Figure 32. Transformation model and the predefined object types

Figure 33 shows the object types that participates in the subject area model.

Predefined DataGuide object types in the sample information catalog

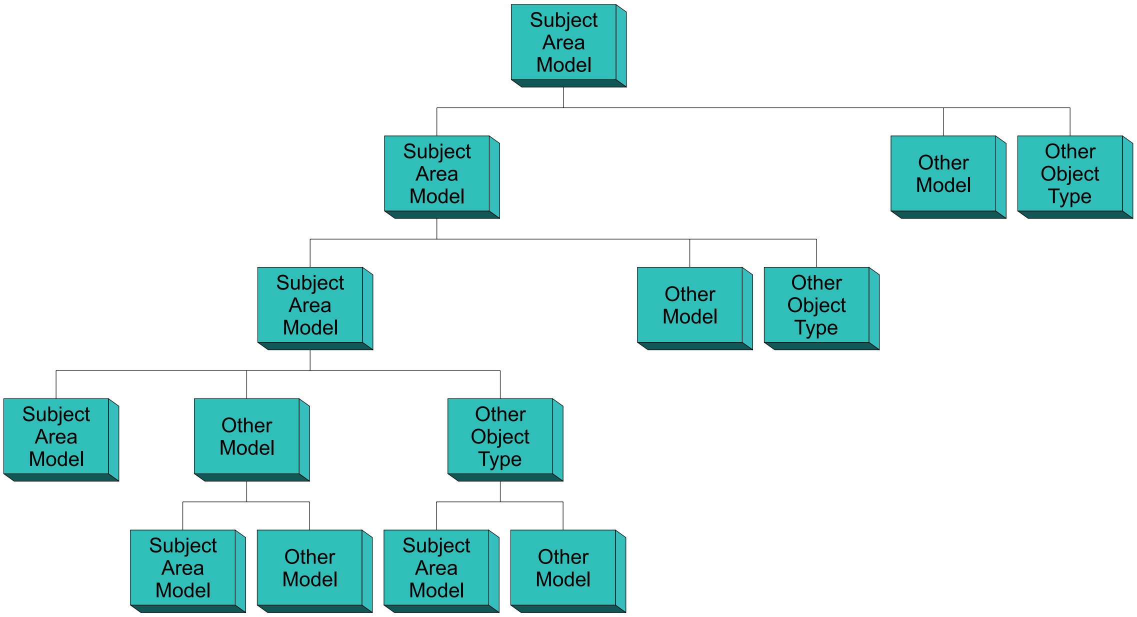


Figure 33. Subject area model and the predefined object types

Predefined object type descriptions

Sample DataGuide object types are organized by category and defined in the tables that start on page 168.

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 DataGuide 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 DataGuide 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 DataGuide object type properties begin on page 61. For each object type that conforms to the Metadata Interchange Specification (MDIS), the MDIS equivalent for each property appears in the column entitled **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 DataGuide 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 DataGuide 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 DataGuide DB2 storage table FLG.OVERDESC.

Grouping category

The Grouping category contains the object types listed in Table 60 on page 168.

Predefined DataGuide object types in the sample information catalog

Table 60. Grouping category object types summary

Object type name	Table name	Description
Application data	XXX.APPLDATA	Used by DataGuide for some MDIS metadata exchanges. Objects of this object type might appear in your information catalog, but you won't use this object type to create objects.Table 61 on page 171 is for reference only. The tag language for defining the "Application data" object type is in the file FLGNYAPL.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 61 on page 171 for a description of properties for this object type.
Business subject areas	XXX.INFOGRPS	Represents logical groupings of objects. The tag language for defining the "Business subject areas" object type is in the file FLGNYINF.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 62 on page 173 for a description of properties for this object type.
Columns or fields	XXX.COLUMNS	Represents columns within a relational table, fields within a file, or fields within an IMS segment. The tag language for defining the Columns or fields object type is in the file FLGNYCOL.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 63 on page 174 for a description of properties for this object type.
Databases	XXX.DATABAS	Represents relational databases. The tag language for defining the "Databases" object type is in the file FLGNYDAT.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 64 on page 176 for a description of properties for this object type.
Dimensions within a multi-dimensional database	XXX.DIMENSION	Represents dimensions within a multi-dimensional database. A dimension is comprised of members. The tag language for defining the "Dimensions within a multi-dimensional database" object type is in the file FLGNYDIM.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 65 on page 178 for a description of properties for this object type.

Table 60. Grouping category object types summary (continued)

Object type name	Table name	Description
Elements	XXX.ELEMENT	Represents MDIS element objects that do not map directly to the Columns or fields object type. The tag language for defining the “Elements” object type is in the file FLGNYELE.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 66 on page 179 for a description of properties for this object type.
Files	XXX.FILE	Represents a file within a file system. The tag language for defining the “Files” object type is in the file FLGNYFIL.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 67 on page 181 for a description of properties for this object type.
IMS database definitions (DBD)	XXX.IMSDBD	Represents IMS database definitions. The tag language for defining the “IMS database definitions (DBD)” object type is in the file FLGNYDBD.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 68 on page 183 for a description of properties for this object type.
IMS program control blocks (PCB)	XXX.IMSPCB	Represents IMS program control blocks. The tag language for defining the “IMS program control blocks (PCB)” object type is in the file FLGNYPCB.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 69 on page 185 for a description of properties for this object type.
IMS program specification blocks (PSB)	XXX.PSB	Represents IMS program specification blocks. The tag language for defining the “IMS program specification blocks (PSB)” object type is in the file FLGNYPSB.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 70 on page 186 for a description of properties for this object type.

Table 60. Grouping category object types summary (continued)

Object type name	Table name	Description
IMS segments	XXX.IMSSEG	Represents IMS segments. The tag language for defining the “IMS segments” object type is in the file FLGNYSEG.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 71 on page 187 for a description of properties for this object type.
Members within a multi-dimensional database	XXX.MEMBER	Represents a member within a multi-dimensional database. A member is part of a dimension, and a dimension is part of a multi-dimensional database. The tag language for defining the “Members within a multi-dimensional database” object type is in the file FLGNYMEM.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 72 on page 189 for a description of properties for this object type.
Multi-dimensional databases	XXX.OLAPMODL	The “Multi-dimensional databases” object type represents multi-dimensional databases. The tag language for defining the “Multi-dimensional databases” object type is in the file FLGNYOLA.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 73 on page 191 for a description of properties for this object type.
Records	XXX.RECORD	The “Records” object type 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 the “Records” object type is in the file FLGNYREC.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 74 on page 193 for a description of properties for this object type.

Table 60. Grouping category object types summary (continued)

Object type name	Table name	Description
Relational tables and views	XXX.TABLES	The “Relational tables and views” object type represents tables or views of relational databases. The tag language for defining the “Relational tables and views” object type is in the file FLGNYTAB.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 75 on page 194 for a description of properties for this object type.
Subschemas	XXX.SUBSCHEM	The “Subschemas” object type represents logical groupings of records within a database. The tag language for defining the “Subschemas” object type is in the file FLGNYSUB.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 76 on page 196 for a description of properties for this object type.
Transformations	XXX.FILTER	The “Transformations” object type 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 the “Transformations” object type is in the file FLGNYFLT.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 77 on page 198 for a description of properties for this object type.

“Application data” object type properties:

Table 61. 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	

Table 61. Properties of the “Application data” 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	
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	

Table 61. Properties of the “Application data” object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Timestamp source definition last changed	CHAR	26	SRCDATCF	O	
Note: S = generated by DataGuide, R = required, O = optional					

“Business subject areas” object type properties:

Table 62. 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	
Note: S = generated by DataGuide, R = required, O = optional					

“Columns or fields” object type properties:

Table 63. 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
Precision of column or field	CHAR	5	PRECDIG	O		ElementPrecision
Can column or field be null	CHAR	1	NULLS	O		ElementNulls

Table 63. 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:
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
Is data a before or after image, or computed	CHAR	50	COLIMAGE	O		ApplicationData

Table 63. 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:
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 DataGuide, R = required, O = optional						

“Databases” object type properties:

Table 64. 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		

Table 64. 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:
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
Agent type	VARCHAR	80	AGENTYPE	O		ApplicationData
Timestamp source definition created	CHAR	26	CRTTIME	O		DateCreated, TimeCreated

Table 64. 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:
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated
Note: S = generated by DataGuide, R = required, O = optional						

“Dimensions within a multi-dimensional database” object type properties:

Table 65. Properties of the “Dimensions within a multi-dimensional 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		
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

Table 65. Properties of the “Dimensions within a multi-dimensional 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:
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
Timestamp source definition last changed	CHAR	26	SRCDATCF	O		DateUpdated, TimeUpdated

Note: S = generated by DataGuide, R = required, O = optional

“Elements” object type properties:

Table 66. 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	

Table 66. 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
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
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	

Table 66. 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
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 DataGuide, R = required, O = optional					

“Files” object type properties:

Table 67. 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		

Table 67. 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:
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
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

Table 67. 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 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 DataGuide, R = required, O = optional						

“IMS database definitions” object type properties:

Table 68. 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

Table 68. 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 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
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

Table 68. 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:
Note: S = generated by DataGuide, R = required, O = optional						

“IMS program control blocks (PCB)” object type properties:

Table 69. 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

Table 69. Properties of the “IMS program control blocks (PCB)” 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:
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 DataGuide, R = required, O = optional						

“IMS program specification blocks (PSB)” object type properties:

Table 70. 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

Table 70. 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:
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
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 DataGuide, R = required, O = optional

"IMS segments" object type properties:

Table 71. 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 71. 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 71. 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 DataGuide, R = required, O = optional

“Members within a multi-dimensional database” object type properties:

Table 72. Properties of the “Members within a multi-dimensional 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		
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

Table 72. Properties of the “Members within a multi-dimensional 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:
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
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

Table 72. Properties of the “Members within a multi-dimensional 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:
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 DataGuide, R = required, O = optional						

“Multi-dimensional databases” object type properties:

Table 73. Properties of the “Multi-dimensional 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

Table 73. Properties of the “Multi-dimensional 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:
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
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 DataGuide, R = required, O = optional						

“Records” object type properties:

Table 74. 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		
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

Table 74. 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:
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 DataGuide, R = required, O = optional						

“Relational tables and views” object type properties::

Table 75. 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

Table 75. 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:
Local database alias	CHAR	8	DBALIAS	O		ApplicationData
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

Table 75. 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:
Transformation program last compiled	CHAR	26	PGMCOMP	O		ApplicationData
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 DataGuide, R = required, O = optional						

“Subschemas” object type properties:

Table 76. 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
Last Changed Date and Time	TIMESTAMP	26	UPDATIME	S		

Table 76. 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 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 DataGuide, R = required, O = optional

“Transformations” object type properties:

Table 77. 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
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

Table 77. 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:
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 DataGuide, R = required, O = optional

Elemental category

The Elemental category contains the object types listed in Table 78.

Table 78. Elemental category object types summary

Object type name	Table name	Description
Audio clips	XXX.AUDIO	The Audio clips object type 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 the Audio clips object type is in the file FLGNYAUD.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 79 on page 201 for a description of properties for this object type.
Charts	XXX.CHARTS	The Charts object type represents either printed or electronic charts. The tag language for defining the Charts object type is in the file FLGNYCHA.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 80 on page 202 for a description of properties for this object type.

Table 78. Elemental category object types summary (continued)

Object type name	Table name	Description
Documents	XXX.DOCS	The Documents object type represents books and technical papers. These publications might be printed or electronic, found locally or within a library. The tag language for defining the Documents object type is in the file FLGNYDOC.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 81 on page 203 for a description of properties for this object type.
Images or graphics	XXX.IMAGES	The Images or graphics object type represents graphic images, such as bitmaps. The tag language for defining the Images or graphics object type is in the file FLGNYIMA.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 82 on page 204 for a description of properties for this object type.
Internet documents	XXX.INTERNET	The Internet documents object type represents Web sites and other documents on the Internet that might be of interest. The tag language for defining the Internet documents object type is in the file FLGNYINT.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 83 on page 205 for a description of properties for this object type.
Lotus Approach queries	XXX.APPROACH	The Lotus Approach queries object type represents Lotus Approach queries for available use with your organization’s data. The tag language for defining the Lotus Approach queries object type is in the file FLGNYAPR.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 84 on page 205 for a description of properties for this object type.
Presentations	XXX.PRESENT	The Presentations object type represents printed or electronic presentations. These presentations might include product, customer, quality, and status presentations. The tag language for defining the Presentations object type is in the file FLGNYPRE.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 85 on page 206 for a description of properties for this object type.

Table 78. Elemental category object types summary (continued)

Object type name	Table name	Description
Spreadsheets	XXX.SSHEETS	The Spreadsheets object type represents desktop spreadsheets (for example, Lotus 1-2-3 or Microsoft Excel spreadsheets). The tag language for defining the Spreadsheets object type is in the file FLGNYSSH.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 86 on page 207 for a description of properties for this object type.
Text-based reports	XXX.REPORTS	The Text-based reports object type represents either printed or electronic reports. The tag language for defining the Text-based reports object type is in the file FLGNYREP.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 87 on page 208 for a description of properties for this object type.
Video clips	XXX.VIDEO	The Video clips object type 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 the Video clips object type is in the file FLGNYVID.TYP in the \VWSWIN\DGWIN\TYPES directory. SeeTable 88 on page 209 for a description of properties for this object type.

“Audio clips” object type properties:

Table 79. 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	

Table 79. Properties of the “Audio clips” object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
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
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by DataGuide, R = required, O = optional					

“Charts” object type properties:

Table 80. 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	

Table 80. Properties of the “Charts” object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Chart publication date	CHAR	26	RPRTDATE	O	
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 DataGuide, R = required, O = optional

“Documents” object type properties:

Table 81. 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	
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	

Table 81. Properties of the “Documents” object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
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 DataGuide, R = required, O = optional					

“Images or graphics” object type properties:

Table 82. 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	
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 DataGuide, R = required, O = optional					

“Internet documents” object type properties::

Table 83. 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 DataGuide, R = required, O = optional					

“Lotus Approach queries” object type properties::

Table 84. 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	

Table 84. Properties of the “Lotus Approach queries” 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	
Approach object filename	VARCHAR	254	FILENAME	R	1
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by DataGuide, R = required, O = optional					

“Presentations” object type properties::

Table 85. 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	

Table 85. Properties of the “Presentations” object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
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 DataGuide, R = required, O = optional					

“Spreadsheets” object type properties:

Table 86. 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	

Table 86. Properties of the “Spreadsheets” 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 DataGuide, R = required, O = optional					

“Text-based reports” object type properties:

Table 87. 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

Table 87. Properties of the “Text-based reports” object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
Report class or type	VARCHAR	80	TYPE	R	2
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by DataGuide, R = required, O = optional					

“Video clips” object type properties:

Table 88. 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 DataGuide, R = required, O = optional					

Contact category

The Contact category contains the People to contact object type. The table name is XXX.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.

"People to contact" object type properties:

Table 89. 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	
Contact's picture filename	VARCHAR	254	FILENAME	O	
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by DataGuide, R = required, O = optional					

Dictionary category

The Dictionary category contains the Glossary entries object type. The table name is XXX.GLOSSARY.

The Glossary entries object type represents definitions for terms used in the information catalog. Its properties are shown in Table 90.

The tag language for defining the Glossary entries object type is in the file FLGNYGLO.TYP in the \VWSWIN\DGWIN\TYPES directory.

“Glossary” object type properties:

Table 90. 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	
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	

Table 90. Properties of the “Glossary entries” 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 DataGuide, R = required, O = optional					

Support category

The Support category contains the object types listed in Table 91.

Table 91. Support category object types summary

Object type name	Table name	Description
DataGuide news	XXX.DGNEWS	The DataGuide news object type contains information about changes to the information catalog. The tag language for defining the DataGuide news object type is in the file FLGNYDGN.TYP in the \VWSWIN\DGWIN\TYPES directory. See Table 92 on page 213 for a description of properties for this object type.
Online news services	XXX.OLNEWS	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. See Table 93 on page 213 for a description of properties for this object type.
Online publications	XXX.OLPUBS	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. See Table 94 on page 214 for a description of properties for this object type.

“DataGuide news” object type properties:

Table 92. Properties of the “DataGuide 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 DataGuide, R = required, O = optional					

“Online news services” object type properties:

Table 93. 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

Table 93. Properties of the “Online news services” 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	
Service name	VARCHAR	254	SERVNAME	R	
URL to access data	VARCHAR	254	URL	O	
Note: S = generated by DataGuide, R = required, O = optional					

“Online publications” object type properties:

Table 94. 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	

Table 94. Properties of the “Online publications” 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 DataGuide, R = required, O = optional					

Program category

The Program category can contain only the Programs object type. The Programs object type is created when an information catalog is created. It is used to define an application that is capable of processing a particular object type.

The table name is XXX.GLOSSARY.

In the sample information catalog, DGV5SAMP, the Programs object type is named “Programs that can be invoked from DataGuide objects”. Its properties are shown in Table 95.

“Programs that can be invoked from DataGuide objects” object type properties:

Table 95. Properties of the “Programs that can be invoked from DataGuide 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

Table 95. Properties of the “Programs that can be invoked from DataGuide objects” object type (continued)

Property name [1]	Data type	Size	Property short name	Value flag [2]	UUI order
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 *Managing DataGuide*.
2. S = generated by DataGuide, 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.

The Comments object type is used to comment on other objects in the information catalog. Its properties are shown in Table 96.

“Comments” object type properties:

Table 96. 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	

Table 96. Properties of the “Comments” object type (continued)

Property name	Data type	Size	Property short name	Value flag	UUI order
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 DataGuide, R = required, O = optional					

Predefined program objects

Program object types shown in Table 97 are provided in the DataGuide sample information catalog. The table also shows the property name you use to associate with the DataGuide program object when launching a program.

Table 97. Generic predefined program objects in the sample information catalog

Type of information	Program name	Object type	Property name
Multi-media 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
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

Table 97. Generic predefined program objects in the sample information catalog (continued)

Type of information	Program name	Object type	Property name
	Microsoft Internet Explorer	Online publications	URL to access data

Table 98 on page 219 lists specific business partners who have applications that are integrated with DataGuide. The information in this table similar to that in Table 97 on page 217.

Table 98. Predefined program objects in sample information catalog — 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	Spreadsheet	Spreadsheet filename
Brio	Brio Query	Text-based report	Report filename
	Netscape Navigator (use with Brio.Insights plug-in)	Text-based report	URL to access data
	Microsoft Internet Explorer (use with Brio.Insights plug-in)	Text-based report	URL to access data
BusinessObjects	BusinessObjects	Databases	None
	BusinessObjects	Report filename	Report filename
	Microsoft Excel (used with BusinessQuery add-in)	Spreadsheets	Spreadsheet filename
	Microsoft Internet Explorer (used to access WebIntelligence Java applet)	Spreadsheets	Spreadsheet filename
	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

Chapter 11. Tag language

The DataGuide tag language allows you to format your descriptive data so that you can import it into DataGuide. The tag language tells DataGuide what to do with the descriptive data that it imports. DataGuide also exports descriptive data into tag language files so that you can back up your DataGuide 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 DataGuide object types and objects. You can also write and use extract programs to extract descriptive data from other sources, such as a relational database management system catalog, that can be imported into DataGuide. Table 99 shows the tags in the DataGuide tag language and the actions that these tags perform.

Table 99. DataGuide tags

Task	Tag names	For details
Record diskette sequence	DISKCNTL	see page 237
Identify action to be taken on input data	ACTION.OBJINST ACTION.OBJTYPE ACTION.RELATION	see page 225 see page 230 see page 234
Describe data to the DataGuide information catalog	OBJECT PROPERTY INSTANCE RELTYPE	see page 244 see page 251 see page 238 see page 255
Identify when changes are committed and where check point occurs	COMMIT	see page 236
Identify user comments	COMMENT	see page 235
Format data	NL TAB	see page 244 see page 256

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 listed in Table 99 on page 221.
- 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_name*s) with the PROPERTY or INSTANCE tags.
- These property names are reserved by DataGuide:

  OBJTYPID

  INSTIDNT

  NAME

  UPDATIME

  UPDATEBY

  You can specify NAME as the *short_name* on the PROPERTY tag if you are identifying 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 DataGuide reads tag language files

When you code a tag language file, consider how DataGuide reads tag language files:

- DataGuide reads the entire tag language file as a continuous data stream.
- DataGuide treats any character with a hexadecimal value under X’20’ (except for tab and new line character tags specified in property values) as a control character and ignores that character.

- DataGuide 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 100 are enabled for double-byte character set (DBCS) support.

Table 100. Keyword values enabled for DBCS

Tag name	Keywords	Variable value
OBJECT	EXTNAME ICOFILE ICWFILE	*ext_name* *OS/2_ICON_file_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.

- DataGuide accepts DBCS blanks only in the keyword values shown in Table 101. If DBCS blanks are found anywhere else in the tag language file, errors can occur.

Table 101. 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 DataGuide descriptive data

Table 102 shows the valid data types for DataGuide descriptive data.

Table 102. Valid data types for DataGuide 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`
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.

DataGuide 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, DataGuide inserts the values shown in Table 103.

Table 103. DataGuide-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 appear in the text. The tags and keywords are represented like this:

```
:tagname.keyword() keyword()
```

Valid values that can be substituted for variables are described in the keyword list. The values are represented like this: *variable*

In tag descriptions, when each pair of keywords or values in a series is separated by a vertical bar, you must include one of the terms 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 following the ACTION tag.

Context

ACTION.OBJINST is used to create, delete, or maintain DataGuide objects.

ACTION.OBJINST is followed by one or more OBJECT and INSTANCE tags, which define the object being acted upon.

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 34. 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 be added.

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 35. 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 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) tag to describe objects of different object types to be deleted.
- If the object being deleted 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 attached to it, and all ATTACHMENT, CONTACT, and LINK relationships in which it participates. Deletes all objects 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 36. 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 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_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 37. 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 DataGuide information catalog to see whether the input object exists.

If the object exists, DataGuide updates the property values of the object in the DataGuide information catalog. If the object does not exist, DataGuide 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 38. Using the ACTION.OBJINST tag when merging objects

Rules:

- If the object exists, DataGuide updates the property values of the object in the DataGuide information catalog. If the object does not exist, DataGuide 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 is because DataGuide must ensure that the object type exists in the Information catalog it is importing into before it can try to add or update (merge) objects.

  You cannot use ACTION.OBJTYPE(MERGE) for an object type belonging 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 39. 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 being 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 be performed on the object type described with the tags following ACTION.OBJTYPE.

Context

ACTION.OBJTYPE is used to create, delete, or maintain DataGuide object types.

ACTION.OBJTYPE is followed by one or more OBJECT and PROPERTY tags, which define the object type being acted upon.

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 40. 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 belonging to the new object type. DataGuide automatically defines the following required properties for every object type:

    OBJTYPID

    INSTIDNT

    NAME

    UPDATIME

    UPDATEBY
- You cannot add object types belonging 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 41. Using the ACTION.OBJTYPE tag when adding properties to object types

Rules:

- The object type must already 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 belonging to the Attachment category.

ACTION.OBJTYPE(DELETE)

Deletes the object type.

Context:

```
:ACTION.OBJTYPE(DELETE)
:OBJECT.TYPE()
```

Figure 42. Using the ACTION.OBJTYPE tag when deleting object types

Rules:

- The object type must already exist. No objects of the object type can exist.
- One or more OBJECT tags must follow ACTION.OBJTYPE(DELETE). Each OBJECT tag identifies the object type being deleted.
- You cannot delete object types belonging 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 43. 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 belonging to the Program or Attachment categories.

ACTION.OBJTYPE(MERGE)

Checks the DataGuide information catalog for the input object type name to see if the object type already exists.

If the object type exists, DataGuide 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 invalid.

If the object type does not exist, DataGuide 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 44. 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 belonging 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 is required earlier in the tag language file than an ACTION.OBJINST(MERGE) tag.
- You cannot merge object types belonging 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 45. 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 be performed on the relationship described with the tags following ACTION.RELATION.

Context

ACTION.RELATION is used to create or delete DataGuide relationships.

ACTION.RELATION is followed by one or more RELTYPE and INSTANCE tags, which define the relationships being acted upon.

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 46. 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, DataGuide 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 being added and specifies the object types of the objects being associated.
  - Each INSTANCE tag specifies the UUI property values that identify the two objects being associated.

ACTION.RELATION(DELETE)

Deletes a relationship.

Context:

```
:ACTION.RELATION(DELETE)
:RELTYPE.TYPE() SOURCETYPE() TARGETTYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 47. Using the ACTION.RELATION tag when deleting relationships

Rules:

- The relationship is deleted if it exists; otherwise, DataGuide 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 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.

DataGuide ignores comments when importing a tag language file.

Syntax

```
:COMMENT.your comments
```

```
:COMMENT.This is the text of a comment.
```

Figure 48. 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 DataGuide tags (for example `:ACTION.`) because each tag is ended by either the end of the file or by the beginning of the next valid DataGuide tag.

COMMIT

Identifies a commit point. Requests that DataGuide commit the current changes to the database.

If DataGuide encounters an error while importing a tag language file, it rolls back all changes made to the DataGuide information catalog since the last time changes were committed.

Include COMMIT checkpoints at regular intervals so that you import DataGuide 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 DataGuide cancels when it rolls back the DataGuide 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, DataGuide clears the echo file of the tags that were processed before the COMMIT tag, so that the echo file only contains tags describing 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 49. Example of a COMMIT tag

Keywords

CHKPID
Required keyword.

checkpt_id
An identifier that DataGuide saves when it processes a COMMIT tag.

If the import of a tag language file fails after a COMMIT tag is processed successfully, you need to import the rest of the tag language file starting at the last checkpoint (an option that is available with the Import function). DataGuide 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 or more diskettes, DISKCNTL is the first tag on each diskette.

Syntax

```
:DISKCNTL.SEQUENCE(nn, + | -)
```

```
:DISKCNTL.SEQUENCE(01,+)
```

Figure 50. 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 indicating 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, so that the numbers for a set of three diskettes is 1, 2, and 3, or 01, 02, and 03

+ Additional diskettes containing the tag language file follow this one.

– The last or only diskette containing 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 fixed disk, this tag is not applicable. If the tag is present, it is ignored.

INSTANCE

Defines or identifies objects or relationships to be acted upon.

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 51. Using the INSTANCE tag when adding objects

```
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE()
:INSTANCE.short_name()
:short_name()
:short_name()
```

Figure 52. 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 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 52.

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 defined as required.
  - You can omit a property that does not have a value to be added 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 DataGuide information catalog. If an omitted required property has a TIMESTAMP data type, then DataGuide 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 DataGuide 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 53. Using the INSTANCE tag when deleting objects

```
:ACTION.OBJINST(DELETE_TREE_ALL)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 54. 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 55. 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 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 defined as a UUI property for the object type. Each object type has one or more properties defined as UUI properties. These properties are used by DataGuide to 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 56. 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 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 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 defined as a UUI property for the object type. Each object type has one or more properties defined as UUI properties. These properties are used by DataGuide to uniquely identify an object in the information catalog.

If you specify a property value, that value is updated in the DataGuide 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 57. Using the INSTANCE tag when adding relationships

```
:ACTION.RELATION(DELETE)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 58. 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 be linked

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 be linked

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 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 defined as a UUI property for the object type.

Each object type has one or more properties defined as UUI properties. These properties are used by DataGuide to 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 59.

```
:INSTANCE.SOURCEKEY(UUIname1(value1) UUIname2(value2)) sname3(value3) sname4(value4)
```

Figure 59. Example of an INSTANCE tag with several short names

Leading blanks included between the parentheses for a value become part of the value; trailing blanks are removed. DataGuide counts these blanks as part of the data length when determining whether the length of the value is valid. Including extra leading or trailing blanks on a value that make the entire value longer than the maximum length for a value causes an error.

NL

Specifies a new line within a property value.

DataGuide only reads NL tags 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(OS/2_ICON_file_name)
           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 60. 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 61. 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 DataGuide information catalogs containing the same object type, so that objects of this object type can be shared among these 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 or embedded blanks are allowed.

After you create the object type, you cannot change the value of *type*.

CATEGORY
Specifies which DataGuide category this object type belongs to.

Required keyword.

category
Specifies a DataGuide 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 modify 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 DataGuide information catalogs.

The value of *ext_name* is stored in mixed case.

You can modify the information on this keyword after the object type is defined.

PHYNAME
Specifies the name used when creating the database table containing information about this object type.

Optional keyword.

table_name
Specifies the name used when creating the database table containing object type information.

The maximum length of the name is defined when DataGuide is installed. *table_name* must be unique within the DataGuide information catalog and cannot contain any SQL reserved words.

By default, *table_name* is the *type* specified for the **TYPE** keyword. This value is not case sensitive; you can specify this value using 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 or embedded blanks are allowed. This value cannot be any of the SQL reserved words for the database used for the DataGuide information catalog.

After the table is created, you cannot change its name.

ICOFILE

Specifies the file containing the OS/2 icon associated with the object type.

Optional keyword.

OS/2_ICON_file_name

Specifies the name of the OS/2 ICON file to be associated with the object type. The maximum length of *OS/2_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 allowable 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 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 *Programming Guide and Reference*), the import function on the user interface or the IMPORT option of the DGUIDE command (see *Managing DataGuide*).

You can modify this value after the object type is created using ACTION.OBJTYPE(UPDATE). However, once you specify an icon file to be associated with an object type, you can change the associated icon, but you cannot redefine the object type to have no associated icon at all.

ICWFILE

Specifies the file containing the Windows icon associated with the object type.

Optional keyword.

Windows_ICON_file_name

Specifies the name of the Windows ICON file to be associated 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 allowable 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 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

Programming Guide and Reference), the import function on the user interface or the IMPORT option of the DGUIDE command (see *Managing DataGuide*).

You can modify this value after the object type is created using ACTION.OBJTYPE(UPDATE). However, once you specify an icon file to be associated with an object type, you can change the associated icon, but you cannot redefine the object type to have no associated icon at all.

ACTION.OBJTYPE(APPEND)

Context:

```
:ACTION.OBJTYPE(APPEND)
:OBJECT.TYPE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 62. 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 63. Using the OBJECT tag when deleting object types

```
:ACTION.OBJTYPE(DELETE_EXT)
:OBJECT.TYPE()
```

Figure 64. 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 65. 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 DataGuide information catalogs.

The value of *ext_name* is stored in mixed case.

ICOFILE
Specifies the file containing the OS/2 icon associated with the object type.

Optional keyword.

OS/2_ICON_file_name
Specifies the name of the OS/2 ICON file to be associated with the object type.

You can update this value.

The maximum length of *OS/2_ICON_file_name* is 254 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, the import function on the user interface, or the IMPORT option of the DGUIDE command.

ICWFILE

Specifies the file containing the Windows icon associated with the object type.

Optional keyword.

Windows_ICON_file_name

Specifies the name of the Windows ICON file to be associated with the object type.

You can update this value.

The maximum length of *Windows_ICON_file_name* is 254 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, the import function on the user interface, or the IMPORT option of the DGUIDE command.

ACTION.OBJINST

Adding, updating, deleting, or merging objects

Context:

```
:ACTION.OBJINST(ADD)
:OBJECT.TYPE()
:INSTANCE.short_name()
```

Figure 66. Using the OBJECT tag when adding objects

```
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE()
:INSTANCE.short_name()
```

Figure 67. Using the OBJECT tag when merging objects

```
:ACTION.OBJINST(UPDATE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) short_name()
```

Figure 68. Using the OBJECT tag when updating objects

```
:ACTION.OBJINST(DELETE)
:OBJECT.TYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...)
```

Figure 69. 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 70. 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 71. Using the PROPERTY tag when merging object types

```
:ACTION.OBJTYPE(APPEND)
:OBJECT.TYPE()
:PROPERTY.EXTNAME() DT() DL() SHRTNAME() NULLS() UUISEQ()
```

Figure 72. 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* 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. *short_name* 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 translated into uppercase.

This value can start with the characters A - Z, @, #, or $, and can contain any of these characters, plus 0 - 9 and _. No leading or embedded blanks are allowed.

This value cannot be any of the SQL reserved words for the database used for the DataGuide information catalog. Do not specify the property short names of any of the following required properties for every DataGuide 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 using 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 is provided for a required property when an object is added to the DataGuide information catalog, DataGuide enters a not-applicable symbol for the required value if it has a data type of CHAR, VARCHAR, or LONG VARCHAR. For a required value with a data type of TIMESTAMP, DataGuide enters the following value: 9999-12-31-24.00.00.000000

UUISEQ

Identifies the properties 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 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 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 DataGuide reserved property NAME as part of the UUI when you add a new object type or merge object types. Figure 73 shows the general syntax for identifying NAME as a UUI property.
  Empty parentheses in this figure denote values that must be provided when used in a tag language file.

```
:ACTION.OBJTYPE(ADD)
:OBJECT.TYPE() CATEGORY() EXTNAME() PHYNAME() ICOFILE() ICWFILE()
:PROPERTY.SHRTNAME(NAME) UUISEQ()
```

Figure 73. Example of specifying the NAME property as part of the UUI

- The maximum length of the UUI fields is 254 bytes.

RELTYPE

Identifies the type of relationship that is being added or deleted 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) TARGETYPE(target_type)
```

Context

```
:ACTION.RELATION(ADD)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 74. Using the RELTYPE tag when adding relationships

```
:ACTION.RELATION(DELETE)
:RELTYPE.TYPE() SOURCETYPE() TARGETYPE()
:INSTANCE.SOURCEKEY(UUI_short_name()...) TARGETKEY(UUI_short_name()...)
```

Figure 75. 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 using 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.

TARGETTYPE
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 using uppercase or lowercase characters.

For an Attachment relationship, *target_type* is the Attachment object type name.

For a Contains relationship, *target_type* is the containee's 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.

DataGuide only reads TAB tags 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 12. What a tag language file should look like

You can use the tags to tell DataGuide to add, delete, and update object types and objects. DataGuide 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 both actions and what you are acting on.

Defining what you want to do

The ACTION tag tells DataGuide what you want to do. The keyword tells DataGuide what kind of information you want to maintain. The option tells DataGuide 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 told DataGuide 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

To define:	**Use these tags:**
New properties for an object type	OBJECT and PROPERTY
New or existing object	OBJECT and INSTANCE
New or existing object relationship	RELTYPE and INSTANCE

Putting it all together

The keywords and values 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 238, “OBJECT” on page 244, or “PROPERTY” on page 251.

Committing changes to the database

The COMMIT tag tells DataGuide to commit changes to the DataGuide database. When DataGuide processes a COMMIT tag, it empties the echo file before it starts processing the next set of tags, so that the echo file only contains tags describing uncommitted changes.

If DataGuide 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 defining the data 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 DataGuide information catalog.

```
:COMMENT.Updating the LASTDATE property
```

Part 3. Supplied program and macro reference

Chapter 13. Supplied Visual Warehouse programs

Visual Warehouse supplies the following programs to support integration with Visual Warehouse:

- vwpexunx
- ISV_Sample

vwpexunx

The vwpexunx program remotely issues a command or runs a program. It supports the operating systems checked in the following table:

Windows NT	**UNIX®**	**OS/2**	**AS/400**
✓	✓		

If you are running the vwpexunx program on Windows NT, the rexecd program must also be running on the workstation.

Parameters

Table 104 shows the parameter list for the vwpexunx program. The list includes the predefined token for a parameter if one exists.

Table 104. 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 - (not-applicable symbol).
6	The remote log (summary) file. If there is no log file, specify - (not-applicable symbol).
7	The remote operating system type. Specify either UNIX or WINNT.
8	The password type. Specify either PasswordNotRequired, EnterPassword, or GetPassword.

Table 104. 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 business view. It 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:

```
vwpexunx tomari labriejj db2cmd \usr\labriejj\db2cmd.err - - UNIX EnterPassword mypass
```

where:

tomari Is the name of the remote host

labriejj
Is the user ID used to access the remote host

db2cmd Is the remote program to run

\usr\labriejj\db2cmd.err
Is the path and name of the remote error file

- Indicates that no remote warning file exists

- Indicates that no remote log (summary) file exists

UNIX Is the remote operating system

EnterPassword
Is the password type

mypass Is 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 105 lists the return codes for the vwpexunx program.

Table 105. 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 has not been 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

Table 105. Return codes for the vwpexunx program (continued)

Return code	Description
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 Visual Warehouse objects from the metadata. It supports the operating systems checked in the following table:

Windows NT	UNIX	OS/2	AS/400
✓			

Table 106 shows the parameter list for the ISV_Sample program.

No predefined tokens exist for the parameters.

Table 106. 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
```

where:

SAMPLE Is the ODBC DSN from which to read metadata

labriejj
Is the user ID used to access the ODBC DSN

mypass Is the password used to access the ODBC DSN

The ISV_Sample program uses the ISV_VWP Visual Warehouse program. Business views call the ISV_VWP program to write the input parameters to an output file.

Chapter 14. Net.Data macros

DataGuide for the Web and Visual Warehouse for the Web use 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, DataGuide for the Web requires a user ID and password by default. You can customize the macros to call your own security program instead.

You can also change the graphics files that are included with DataGuide for the Web and Visual Warehouse for the Web. For example, you can add your company logo or a message to your users in the graphics.

This chapter lists the files that are included with DataGuide for the Web and Visual Warehouse for the Web. For more information about Net.Data and its macros, see the *Net.Data Programming Guide* and *Net.Data Reference Guide*.

DataGuide for the Web files

The DataGuide for the Web files are in the DGWEB\ENU directory of the Visual Warehouse CD-ROM.

The file names are lowercase to follow the AIX naming convention.

Table 107 lists the DataGuide for the Web files that contain Net.Data macros, which are located in the DGWEB\ENU\MACRO directory of the Visual Warehouse CD-ROM.

Table 107. DataGuide 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 home page

Table 107. DataGuide Web Net.Data macros (continued)

File name	Description
dg_tableviewer.mac	Displays sample data

Table 108 lists the DataGuide for the Web files that contain Net.Data include files, which are located in the DGWEB\ENU\MACRO directory of the Visual Warehouse CD-ROM.

Table 108. DataGuide Web 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 home page
dg_strings.hti	Include file with translatable strings
dg_config.hti	Include file with installation configurable variables

Table 109 lists the DataGuide for the Web files that contain HTML, which are located in the DGWEB\ENU\HTML directory of the Visual Warehouse CD-ROM.

Table 109. DataGuide for the Web HTML files

File Name	Description
*.htm	Help files

Table 110 lists the DataGuide for the Web graphic files, which are located in the DGWEB\ENU\ICONS directory of the Visual Warehouse CD-ROM.

Table 110. DataGuide for the Web graphics files

File name	Description
dg_bsearch.gif	Search banner
dg_bsearchres.gif	Search results banner
dg_bsubjects.gif	Subjects banner
dg_btree.gif	Tree banner
dg_badvsearch.gif	Advanced search banner
dg_bdesc.gif	Description banner
dg_blongprop.gif	Long property value banner
dg_bcomdesc.gif	Comment description banner

Table 110. DataGuide for the Web graphics files (continued)

File name	Description
dg_bcreatecom.gif	Create comment banner
dg_bupdatecom.gif	Update comment banner
dg_bcopycom.gif	Copy comment banner
dg_sbcomment.gif	Comments sub-banner
dg_sbcontact.gif	Contacts sub-banner
dg_sblink.gif	Linked objects sub-banner
dg_sbtrans.gif	Transformations sub-banner
dg_sbattach.gif	Attached to sub-banner
dg_sbparent.gif	Parent objects sub-banner
dg_lhelp.gif	Help link
dg_lsubjects.gif	Subjects link
dg_lhome.gif	Home link
dg_ladvsearch.gif	Advanced search link
dg_lcomment.gif	Create/update/delete comment link
dg_lbullet.gif	Bullet
dg_lmore.gif	More
dg_lexpand.gif	Expand
dg_grph.gif	Home page logo
dg_psearch.gif	Search push button
dg_clear.gif	Clear image file for spacing

Visual Warehouse for the Web files

The Visual Warehouse for the Web files are in the VWWEB\ENU directory of the Visual Warehouse CD-ROM.

The file names are lowercase to follow the AIX naming convention.

Table 111 on page 274 lists the Visual Warehouse for the Web files that contain Net.Data macros, which are located in the VWWEB\ENU\MACRO directory of the Visual Warehouse CD-ROM.

Table 111. Visual Warehouse for the Web Net.Data macros

File name	Description
vw_operations.mac	Displays the WIP Filter, Operations Work in Progress, Log Viewer, and Log Viewer Details, Statistics, and Statistics Details pages

Table 112 lists the Visual Warehouse for the Web files that contain Net.Data include files, which are located in the VWWEB\ENU\MACRO directory of the Visual Warehouse CD-ROM.

Table 112. Visual Warehouse for the Web Net.Data include files

File Name	Description
vw_strings.hti	Include file with translatable strings
vw_config.hti	Include file with installation configurable variables

Table 113 lists the Visual Warehouse for the Web files that contain HTML, which are located in the VWWEB\ENU\HTML directory of the Visual Warehouse CD-ROM.

Table 113. Visual Warehouse for the Web HTML files

File name	Description
vw_home.html	Sample HTML file to invoke Net.Data macros
*.htm	Help files

Table 114 lists the Visual Warehouse for the Web graphics files, which are located in the VWWEB\ENU\ICONS directory of the Visual Warehouse CD-ROM.

Table 114. Visual Warehouse for the Web graphics files

File name	Description
vw_scheduled.gif	Operations Work in Progress icon for a scheduled business view
vw_successful.gif	Operations Work in Progress icon for a successful business view
vw_canceled.gif	Operations Work in Progress icon for a canceled business view
vw_populating.gif	Operations Work in Progress icon for a populating business view
vw_failed.gif	Operations Work in Progress icon for a failing business view

Table 114. Visual Warehouse for the Web graphics files (continued)

File name	Description
vw_purging.gif	Operations Work in Progress icon for purging a business view
vw_canceling.gif	Operations Work in Progress icon for canceling a business view
vw_retrying.gif	Operations Work in Progress icon for retrying a business view
vw_warning.gif	Operations Work in Progress icon for a warning business view
vw_info.gif	Log Viewer icon for information
vw_warn.gif	Log Viewer icon for warning
vw_stop.gif	Log Viewer icon for stop
vw_lhelp.gif	Help link

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 those tokens that have a specific list of allowed values, circle one of the allowed values.

Table 115. Tokens for required metadata in the templates

Token	Value
**AgentSite*	
**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
**AgentSiteTCPIPHostName*	
**AgentSiteUserid*	
**BVContact*	
**BVCreateTargetTable*	One of the following values: **ISV_CREATETABLEYES** Visual Warehouse is to create the target table. **ISV_CREATETABLENO** Visual Warehouse is not to create the target table.

Table 115. Tokens for required metadata in the templates (continued)

Token	Value
**BVDataNotPresent*	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.
**BVDescription*	
**BVExternalPopulation*	One of the following values: **ISV_BVEXTERNALYES** An external application can populate the table. **ISV_BVEXTERNALNO** Only Visual Warehouse can populate the table.
**BVName*	
**BVNotes*	
**BVSelectStatement*	
**BVSelectStatementGenerated*	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.

Table 115. Tokens for required metadata in the templates (continued)

Token	Value
**BVSQLWarning*	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 fail if an SQL warning code is issued. **ISV_BVSQLWarning_Error** The business view is to process with a warning if an SQL warning code is issued.
**BVType*	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.
**ColumnAllowsNulls*	One of the following values: **ISV_NULLSYES** The column allows null data. **ISV_NULLSNO** The column does not allow null data.
**ColumnDescription*	

Table 115. 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.
**ColumnKeyPosition*	
**ColumnLength*	
**ColumnName*	

Table 115. 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 115. 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*	
**ColumnPositionNumber*	
**ColumnPrecision*	
**ColumnUserActions*	
**ConcurrentBVName*	
**CurrentCheckPointID++*	
**DatabaseContact*	
**DatabaseDescription*	
**DatabaseName*	
**DatabaseNotes*	
**DatabasePhysicalName*	

Table 115. Tokens for required metadata in the templates (continued)

Token	Value
Token	**Value**
Token	**Value**
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 for OS/400® for CISC machines **ISV_IR_DB2400RISC** DB2 for OS/400 for RISC machines **ISV_IR_FFLanLocalCmd** Local flat file **ISV_IR_FFLanFTPCopy** Local flat file sent using FTP from a remote system
**DatabaseServerName*	
**DatabaseUserid*	
**PostBVName*	
**SecurityGroup*	`ISV_DEFAULTSECURITYGROUP`
**SubjectArea*	

Table 115. Tokens for required metadata in the templates (continued)

Token	Value
**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.
**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_CONTAINS_NAMES** The first row of the file contains data.
**TableFullName*	
**TableNotes*	
**TableOwner*	
**TablePhysicalName*	

Table 115. Tokens for required metadata in the templates (continued)

Token	Value
**Token*	
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*	
**VWPInstanceNotes*	
**VWPProgramInstanceKey*	
**VWPProgramInstanceParameterData*	
**VWPProgramInstanceParameterKey*	
**VWPProgramInstanceParameterName*	
**VWPProgramInstanceParameterOrder*	
**VWPProgramTemplateDescription*	
**VWPProgramTemplateExecutableName*	
**VWPProgramTemplateFunctionName*	
**VWPProgramTemplateName*	
**VWPProgramTemplateNotes*	

Table 115. Tokens for required metadata in the templates (continued)

Token	Value
**VWPProgramTemplateType*	One of the following values: **ISV_PROGRAMTYPEDLL** The Visual Warehouse program is loaded from a dynamic load library (DLL) or is a load module. **ISV_PROGRAMTYPECOMMAND** The Visual Warehouse program is a command file. **ISV_PROGRAMTYPEEXECUTABLE** The Visual Warehouse program is an executable file.
**VWPProgramTemplateParameterData*	
**VWPProgramTemplateParameterKey*	
**VWPProgramTemplateParameterName*	
**VWPProgramTemplateParameterOrder*	

Appendix B. Writing your own program for use with Visual Warehouse

You can write Visual Warehouse programs in any language that supports one of the following program types: executable, batch program, or dynamic load library.

If the Visual Warehouse program has a program type of executable, batch program, batch command file, or dynamic load library, it must reside on the agent site. The Visual Warehouse agent starts the program at the scheduled time. On Windows NT, 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 Visual Warehouse 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 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 Visual Warehouse programs that use Object REXX, complete the following procedure to enable these programs to run under Windows NT:

1. Define the Visual Warehouse agent or server service as a system process that can interact with the Windows NT 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 prompt.

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, Visual Warehouse generates a command-line parameter list that it passes as input to your Visual Warehouse program. Whenever possible, test your program from the command prompt before using it in a business view.

Example: The Visual Warehouse program DB2 UDB 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 76:

```
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 76. Reading parameters from the command line

The program uses the target parameters to connect to the target database, as shown in Figure 77 on page 291:

```
rc = SQLConnect (hdbc, (SQLCHAR *)dbName, SQL_NTS,
          (SQLCHAR *)dbUser, SQL_NTS,       /* UID */
          (SQLCHAR *)dbPassword, SQL_NTS);  /* Password */
```

Figure 77. Connecting to the target database

The program then uses the DB2 load utility to load data into the database.

You can see the complete listing of the DB2 UDB load replace program in the VWSAMPLE\VWP directory of the Visual Warehouse CD-ROM.

Returning status information

After your Visual Warehouse program runs, it must return a return code to the business view that uses the program. The return code must be a positive integer. If your program does not return a return code, the business view using the program fails. Visual Warehouse displays the return code in the **Error RC2** field of the Log Details window when the value of **Error RC1** is 8410.

Your Visual Warehouse program can return additional status information to Visual Warehouse:

- Another return code, which can be the same as or different from the code that is returned by the Visual Warehouse program.
- A warning flag that indicates that Visual Warehouse is to treat the return code as a warning. When the Visual Warehouse program sets this flag, the business view that uses this program will have **Warning** status in the Operations Work in Progress window.
- A message, which Visual Warehouse will display in the **System Message** field of the Log Viewer Details window.
- The number of rows of data that the Visual Warehouse program processed.

  Visual Warehouse will display the number in the Log Viewer Details window for the business view.
- The number of bytes of data that the Visual Warehouse program processed.

  Visual Warehouse will display the number in the Log Viewer Details window for the business view.
- The SQLSTATE return code, which Visual Warehouse will display in the SQL state field of the Log Viewer Details window.

The Visual Warehouse agent transfers the additional status information to the Visual Warehouse server.

Transferring the information to Visual Warehouse

To transfer the additional status information to the Visual Warehouse agent, your Visual Warehouse 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 Visual Warehouse program. After the Visual Warehouse 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 Visual Warehouse program cannot create the file, it should continue to run.

Format of the feedback file

Your Visual Warehouse 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 item returned 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.

Visual Warehouse system message
<MSG>*message text*\n</MSG>, where

message text
Is the text of one or more messages

\n Is 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 78 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 78. Example of the feedback file

How the feedback file determines the business view status

The return codes and business view status for the Visual Warehouse 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 Visual Warehouse program returned
- Whether a feedback file exists
- The value of the return code in the feedback file
- Whether the warning flag is set on

Table 116 lists the possible combinations of these values and the results that they produce.

Table 116. Feedback file conditions and results

				Results	
Conditions				**Business view status**[1]	**Values of Error RC1 and RC2**
Visual Warehouse 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-zero[3]	<WARNING> is not set in the feedback file	Failed	**RC1** = 8410 (the Visual Warehouse 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 116. Feedback file conditions and results (continued)

				Results	
Conditions				**Business view status**[1]	**Values of Error RC1 and RC2**
The Visual Warehouse program return code is nonzero	No feedback file exists[2]			Failed	**RC1** = 8410 (the Visual Warehouse program failed); **RC2** = the code returned by the Visual Warehouse 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-zero	<WARNING> is not set in the feedback file	Failed	**RC1** = 8410 (the Visual Warehouse program failed); **RC2** = the code returned by the Visual Warehouse program
			The value of <WARNING> in the feedback file is 1	Warning	**RC1** = 0 **RC2** = the value of <RC> in the feedback file

Notes:

1. Visual Warehouse displays the business view processing status in the Operations Work in Progress window.
2. Visual Warehouse checks for the existence of the feedback file, regardless of whether the return code for the Visual Warehouse program is 0 or nonzero.
3. Visual Warehouse always displays the value of <RC> in the feedback file as the value of the **RC2** field in the Log Details window.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter 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 10594-1785
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
AS/400
DataGuide
DB2
DB2 Connect
DB2 OLAP Server
DB2 Universal Database
IBM
IMS
Net.Data
OS/2
OS/390
OS/400

1–2–3 and Lotus are trademarks of Lotus Development Corporation in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Bibliography

For questions about how Visual Warehouse works, see the online help. Visual Warehouse provides help for specific windows and for general tasks, such as creating information resources and business views.

For information about IBM products that are related to Visual Warehouse, visit the IBM data management Web site at http://www.software.ibm.com/data/

The Visual Warehouse library includes the following publications:

IBM Visual Warehouse for Windows NT: Planning and Installing Visual Warehouse and DataGuide, SC26-3496

IBM Visual Warehouse for Windows NT: Messages and Reason Codes (HTML book that is included in the Visual Warehouse folder)

IBM Visual Warehouse for Windows NT: Installing and Using the Visual Warehouse AS/400 Agent, SC26-9468

IBM Visual Warehouse for Windows NT: Managing DataGuide, SC26-3362

IBM DataGuide: Programming Guide and Reference, SC26-3368

IBM Visual Warehouse for Windows NT: Managing ETI•EXTRACT Conversion Programs with Visual Warehouse, SC26-9467

IBM DB2 OLAP Server: Using DB2 OLAP Server, SC26-9235

Index

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.