IBM® DB2® Warehouse Manager

Information Catalog Manager Programming Guide and Reference

Version 7

SC26-9997-00

IBM® DB2® Warehouse Manager

Information Catalog Manager Programming Guide and Reference

Version 7

SC26-9997-00

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 365.

Contents

About this book

This book is intended for programmers who plan to write applications that work with the Information Catalog Manager. These programs can use application program interface (API) calls to access the Information Catalog Manager functions.

This book assumes that you are familiar with the concepts explained in the *Information Catalog Manager Administration Guide* and with C language programming. You should also have Microsoft Visual C++ Compiler installed.

The Information Catalog Manager provides the application program interface (API) and import/export interfaces for information catalogs.

What is an information catalog?

An information catalog is a mechanism for storing descriptive details, or metadata, about an organization's information resources. An information catalog can help users find what data is available to them and what that data means. When users find data they want, they can use informational applications to retrieve and analyze the data. The Information Catalog Manager provides functions that let users use informational application functions, such as Lotus 1-2-3.

What are information catalog architected interfaces?

This book includes definitions for information catalogs. These interfaces include:

- Application program interface (API)

  The syntax and specifications for input and output for the Information Catalog Manager API calls are documented in "Chapter 5. The Information Catalog Manager API call syntax" on page 67.

  Specifications for the input and output structures used with these API calls are documented in "Chapter 4. The Information Catalog Manager input and output structures" on page 31.

- Import/export interface to the information catalog

  The syntax and information about using the tag language is documented in the *Information Catalog Manager Administration Guide*.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other Data Warehouse Center

publications. Send your comments from the Web. Visit the Web site at http://www.software.ibm.com/data/vw/

The Web site has a feedback page that you can use to enter and send comments.

Chapter 1. Introduction to the Information Catalog Manager

The Information Catalog Manager helps business professionals locate data anywhere in an organization quickly and easily. Users actually access the data using informational applications—applications that allow them to retrieve and analyze their data, without knowing or caring where the data is actually stored.

The Information Catalog Manager helps you learn:
- What data is available
- What the data means in business terms
- Where the data is located
- How you can access the data
- Who you can contact about the data

This information about data is called *descriptive data*, or *metadata*, and is stored in an *information catalog*. Each information catalog is stored in a database that is maintained by the Information Catalog Manager.

Each information source or group of information sources is represented as an *object*. You can use many types of objects to represent the various kinds of information sources your organization uses, such as database tables, spreadsheets, and digitized photographs. From many of these objects, you can start programs that can work with the information sources.

Each information catalog object is similar to a card in a card catalog. Each object provides details about the information source, such as the name of the information source, a description, and the date on which the information source was last updated.

Who uses the Information Catalog Manager?

There are three types of Information Catalog Manager users:
- Users
- Administrators
- Application programmers

Users

In your organization, users make business decisions and contribute to decisions using information they locate using the Information Catalog Manager. Although they might be familiar with various software programs, they do not need to understand database or computer programming concepts.

Who uses the Information Catalog Manager?

Some Information Catalog Manager users can perform additional object management tasks that are normally performed by the Information Catalog Manager administrators if they have been granted authority by their administrator.

Administrators

Administrators manage the Information Catalog Manager. They provide the metadata that helps users locate the data they need. Administrators ensure that the information catalog metadata is available, easy to find and use, current, and protected from unauthorized access.

Application programmers

Application programmers write programs that support information catalog users. The Information Catalog Manager provides C language API calls that let your programs use the Information Catalog Manager functions.

Application programmers need detailed information about how the Information Catalog Manager organizes and stores metadata. See "Chapter 2. Managing objects with an application" on page 5 for information about how an Information Catalog Manager application works with objects.

What kinds of applications work with the Information Catalog Manager?

You can write two types of applications that use the Information Catalog Manager functions:

- Applications that present data to the user
- Tools that help the administrator perform tasks such as adding and updating metadata—extract programs, for example

Informational applications

You can write applications that work with the Information Catalog Manager in two ways. These applications can:

- Start the application from the Information Catalog Manager

  Figure 1 on page 3 shows that users can find the object they want, then start a familiar informational application, running under DOS or Microsoft® Windows®, that works with the information source identified by this object. The Information Catalog Manager passes the necessary metadata to this application.

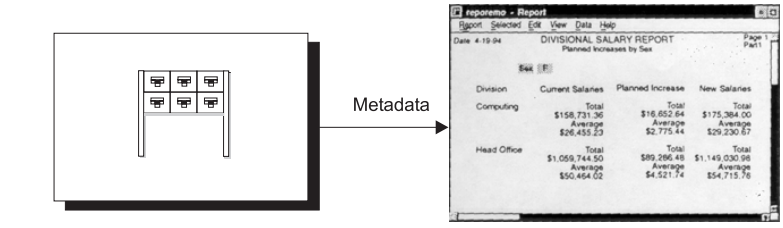

Figure 1. Starting an application from the Information Catalog Manager

- Provide the application with metadata

  Users work with familiarinformational applications that run on DOS or Microsoft Windows. These applications can use the Information Catalog Manager functions to locate the information sources that the user wants to work with. Then these applications can retrieve and analyze the actual data located by the Information Catalog Manager, and present the results to the user using its own user interface, as shown in Figure 2.

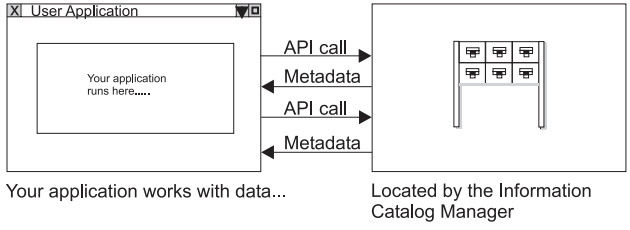

Figure 2. Using an application that lets the Information Catalog Manager locate the data

Tools that maintain and administer the information catalog metadata

You can write tools for your administrator that:

- Maintain the information catalog metadata
- Add metadata to the information catalog

Maintaining an information catalog

One of the main tasks of a administrator is to update the metadata in the information catalog when the information source itself changes. For example, metadata about a file can include the date of the most recent update; if the file is again updated, the administrator needs to update the corresponding date in the metadata.

You can automate this process by writing programs that update the metadata when the corresponding information source changes.

Adding new objects

When administrators create new information catalogs or add newinformation sources to existing information catalogs, they need to add new object types and objects. The administrator can add metadata by importing files that contain the Information Catalog Manager *tag language*. This tag language defines the meaning of the metadata being imported into an information catalog.

You can write applications that automatically generate tag language files based on information specified by the administrator. These files can then be imported into one or more information catalog databases to populate the information catalog with metadata.

You can also write applications that extract metadata from existing data sources and format the data as tag language files. These applications are called *extract programs*, and are described in the *Information Catalog Manager Administration Guide*.

Chapter 2. Managing objects with an application

When you write applications that manage or access metadata in the information catalogs, you need more information about how the information catalog organizes and controls the metadata it stores. This chapter describes:

- How categories, object types, and object instances organize your information catalog
- The two parts of the object type definition
- How to define new object types
- Terminology available for different levels of information catalog users

Organizing objects using categories

The Information Catalog Manager provides seven *categories* for classifying your metadata. These categories control how objects work together to provide a structure for the metadata in your information catalog database. Except for the Program and Attachment categories, you can create new object types in any of the following information catalog categories:

Category	Definition
Grouping	Object types that can contain other object types.
Elemental	Non-Grouping object types that are the building blocks for other Information Catalog Manager object types.
Contact	Object types that identify a reference for more information about an object. More information might include the 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 the Information Catalog Manager object types. The only object type belonging to the Program category is the Programs object type, which is defined when you create an information catalog.
Dictionary	Object types that define terminology that is specific to your business.
Support	Object types that provide additional information about your information catalog or enterprise.
Attachment	A Comments object type that identifies additional information attached to another Information Catalog Manager object. The

only object type belonging to the Attachment category is the Comments object type, which is defined when you create an information catalog.

Table 1 summarizes the relationships among the Information Catalog Manager's object type categories.

Table 1. The Information Catalog Manager category relationships

Category	Can contain/ 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

The Information Catalog Manager lets you organize data about your information sources by defining *object types* and objects.

You use object types to classify your objects. For example, if you have several database tables, you can create an object type for tables so that you can store and maintain similar metadata for each table. For most categories, you can define your object types to contain whatever metadata is most useful for your organization.

Objects contain the metadata for a specific unit of information; for example, information about a table, a person, or a program. An object type is a template for an object; it defines the metadata that you need to store in the information catalog for each similar unit of information. Therefore, consider objects as *instances* of the object type; you can define several instances based on a single object type.

For more information about using different categories to design your information catalog, see the *Information Catalog Manager Administration Guide*.

A programmer's view of the Information Catalog Manager object types

The administrator managing an object type with the user interface or tag language is aware only of working with an object type. However, when you write a program using the Information Catalog Manager API calls to manage an object type, you need to be aware that there are two parts of the object type: the object type registration and the object type itself.

Object type registration
The object type registration contains overall information about the object type, including:

- Category the object type belongs to
- Extended (NAME) and short (DPNAME) names of the object type
- Name of the information catalog database table containing the object instance information

When you create or update the object type registration, you also give the Information Catalog Manager the name of an icon file associated with the object type.

Object type
The object type defines the properties that are used for each object. These properties, such as OWNER and DESCRIPTION, contain information about the information source described by each object.

The above two parts require separate maintenance functions, which are provided by the following the Information Catalog Manager API calls:

For object type registration:	For object type:	Purpose
FLGCreateReg	FLGCreateType	Define a new object type or object type registration
FLGGetReg	FLGGetType	Get information about an object type or object type registration
FLGUpdateReg	FLGAppendType	Change the definition of an object type or object type registration
FLGDeleteReg	FLGDeleteType FLGDeleteTypeExt	Delete an object type or object type registration

When you create or delete an object type, you need to use the FLGCreateReg and FLGCreateType calls or FLGDeleteType and FLGDeleteReg calls as pairs to make sure that complete object types are created or deleted. Object type registrations that do not have associated object types with defined properties

are useless and can cause problems if you later try to use these object types to define objects in your information catalog.

You cannot change or delete object type properties after you create the object type; you can only append new optional properties using the FLGAppendType call (see "FLGAppendType" on page 69).

Defining object types

When defining a new object type, at a minimum you must specify the following:

- Registration properties
- The category the object type belongs to
- Required properties common to all objects

After you complete the above steps, you can define additional optional properties for the object type.

Specifying registration properties

When you register an object type, you must specify these six properties in the order shown in Table 2.

Table 2. Properties required for object type registrations

Position	Property short name	Property name[1]	Description	Comments
1	NAME	EXTERNAL NAME OF OBJ TYPE	80-byte name of the object type.	You must set this value using the FLGCreateReg call. You can modify this value using the FLGUpdateReg call.
2	PTNAME	PHYSICAL TYPE NAME	30-character name of the table in the information catalog database that contains the object type.	You can only set this value using the FLGCreateReg call. You cannot modify this value after the object type is registered.

Table 2. Properties required for object type registrations (continued)

Position	Property short name	Property name[1]	Description	Comments
3	DPNAME	DP NAME	8-character short name for the object type.	You must set this value using the FLGCreateReg call. You cannot modify this value after the object type is registered.
4	CREATOR	CREATOR	8-character user ID of the administrator who creates the object type.	The Information Catalog Manager sets this value when the FLGCreateType call is issued for the object type. You cannot set or modify this value.
5	UPDATEBY	LAST CHANGED BY	8-character user ID of the administrator who last modified the object type.	The Information Catalog Manager sets and modifies this value when the FLGAppendType call is issued to add optional properties to the object type.
6	UPDATIME	LAST CHANGED DATE AND TIME	26-character time stamp of the last date and time the object type was modified.	The Information Catalog Manager sets and modifies this value when the FLGCreateType or FLGAppendType call is issued for the object type.

Note:

1. The property names in this column apply to English versions of the Information Catalog Manager; if you are using a translated version of the Information Catalog Manager, the property name will also be translated.

Specifying the category for a new object type

You set the category of the object type when you register the object type using FLGCreateReg.

Defining object types

You can create object types belonging to the following categories:
- Grouping
- Elemental
- Contact
- Dictionary
- Support

These five categories are briefly described in "Organizing objects using categories" on page 5. For more detailed information, see the *Information Catalog Manager Administration Guide*.

The Information Catalog Manager defines both a Programs and Comments object type when you create a new information catalog database. Programs is the only object type that can belong to the Program category; you cannot create any other Program object types. Comments is the only object type that can belong to the Attachment category; you cannot create any other Attachment object types.

Defining required object type properties

When you define a new object type, you must specify the five required properties shown in Table 3 as the first five properties for the object type. The Information Catalog Manager uses the property short names to identify the required properties.

Table 3. Properties required for every object type

Position	Property short name	Property name	Description	Comments
1	OBJTYPID	Object type identifier	6-character system-generated ID for the object type	The Information Catalog Manager generates a unique identifier for each object type. This value is the first part of the FLGID that you use with several API calls to identify object instances. You cannot modify this value.

Table 3. Properties required for every object type (continued)

Position	Property short name	Property name	Description	Comments
2	INSTIDNT	Instance identifier	10-character system-generated ID for the object instance	The Information Catalog Manager generates a unique identifier for each object instance. This value is the second part of the FLGID that you use with several API calls to identify object instances. You cannot modify this value.
3	NAME	Name	80-byte user-specified name for the object.	This name is displayed by the Information Catalog Manager. You can modify this value using the FLGUpdateInst call.
4	UPDATIME	Last Changed Date and Time	26-character time stamp of the last date and time the object instance was modified.	The Information Catalog Manager sets this value when the object instance is created or modified (using FLGCreateInst or FLGUpdateInst calls). You cannot modify this value.

Table 3. Properties required for every object type (continued)

Position	Property short name	Property name	Description	Comments
5	UPDATEBY	Last Changed By	8-character user ID of the person who last modified the object instance.	The Information Catalog Manager sets and modifies this value when the object instance is created or modified (using FLGCreateInst or FLGUpdateInst calls).

The property short names for these required properties are reserved. Do not use these names for any other property short name assignments.

When you create a new object instance, you must specify a value for NAME. The Information Catalog Manager generates the values for OBJTYPID, INSTIDNT, UPDATIME, and UPDATEBY. You cannot modify these system-generated values.

Identifying your new object type and object instances

When the system generates OBJTYPID, you use this value to uniquely identify a registered and defined object type.

When the system generates INSTIDNT, you use this value with OBJTYPID to uniquely identify a single object instance.

This book refers to the combined OBJTYPID and INSTIDNT values as *FLGID* in "Chapter 5. The Information Catalog Manager API call syntax" on page 67.

The Information Catalog Manager identifier names

Because the Information Catalog Manager is designed to be used by several different levels of users, different terminology is used for describing object types for different product users. You find less technical, more business-oriented terms in the Information Catalog Manager interface and in the books *Information Catalog Manager User's Guide* and the *Information Catalog Manager Administration Guide*.

In this book terms are oriented to the data processing environment for administrators and application programmers.

You need to be aware of these terminology differences when writing applications for users or administrators. Table 4 provides a quick reference to the different levels of terminology.

Table 4. The Information Catalog Manager terminology for object types

Description	User term	Administrator term	Tag language term	API call term
Long (80-byte) name of object type	Object type	Object type name	EXTNAME(*ext_name*)	EXTERNAL NAME OF OBJ TYPE NAME property in the input or output structure
Short (8-character) name of the object type	—	Short name	TYPE (*type*)	DP NAME DPNAME property in the input or output structure
Name of the information catalog database table containing the object type information	—	—	PHYNAME (*table_name*) TYPE (*type*) if PHYNAME is not specified	PHYSICAL TYPE NAME PTNAME property in input or output structure
Long (80-byte) property name	Property	Property name	EXTNAME (*ext_name*)	Property name
Property short (8-character) name	—	Short name	SHRTNAME (*short_name*)	Property short name

Chapter 3. Writing programs with the Information Catalog Manager API calls

The Information Catalog Manager provides C language API calls that let your programs use the Information Catalog Manager functions.

This chapter describes:
- What Information Catalog Manager functions you can perform using API calls
- The general structure of API calls
- How to pass data to and from the Information Catalog Manager API calls
- C language header files provided by the Information Catalog Manager
- How to write a C language program using the Information Catalog Manager
- Rules for using the Information Catalog Manager API calls
- The DG2SAMP.C sample program

What you can do with the Information Catalog Manager API calls

The Information Catalog Manager API calls have consistent syntax rules. See "Chapter 5. The Information Catalog Manager API call syntax" on page 67 for the complete syntax for each API call.

These API calls use self-defining input and output structures. Any programming language can read and generate these structures. For more information about the input structures and output structures, see "Chapter 4. The Information Catalog Manager input and output structures" on page 31.

This section briefly describes all of the API calls provided by the Information Catalog Manager and tells you where to find detailed information about each call.

Provide the Information Catalog Manager application support

These API calls allow your program to use other the Information Catalog Manager API calls.

API call	Purpose	See:
FLGInit	Allocate required resources and initialize the Information Catalog Manager client	142
FLGFreeMem	Free output structures defined by the Information Catalog Manager.	125

API call	Purpose	See:
FLGTerm	Relinquish resources and terminate the Information Catalog Manager client	223
FLGTrace	Set the level of tracing	225

Manage object type registrations

Registrations uniquely identify object types to the Information Catalog Manager.

API call	Purpose	See:
FLGCreateReg	Register a new object type	84
FLGDeleteReg	Delete an object type registration	100
FLGGetReg	Get the information for an object type registration	131
FLGUpdateReg	Update the information for an object type registration	233
FLGManageIcons	Create and update icons that represent an object type	182

Manage object types

Object types define associated properties.

API call	Purpose	See:
FLGAppendType	Add new properties to an object type	69
FLGCreateType	Create a new object type	91
FLGDeleteType	Delete an object type	107
FLGDeleteTypeExt	Delete an object type along with its instances and object type registration	110
FLGGetType	Get information about an object type	135

Manage object instances

Object instances contain metadata representing a unit of information.

API call	Purpose	See:
FLGCreateInst	Create a new object instance	78
FLGDeleteInst	Delete an object instance	97

API call	Purpose	See:
FLGDeleteTree	Delete a Grouping object instance and optionally delete all underlying instances	102
FLGUpdateInst	Update information about an object instance	228
FLGGetInst	Get information about an object instance	127

Manage the Information Catalog Manager identifiers

This API call allows your program to convert identifiers for performance purposes.

API call	Purpose	See:
FLGConvertID	Convert object type and instance identifiers for application performance	76

Define object relationships

Relationships define the interaction of two object instances.

API call	Purpose	See:
FLGRelation	Create or delete a contains, contact, attachment, or link relationship between two object instances.	204

Locate object instances

You can locate object instances based on the values of certain properties.

API call	Purpose	See:
FLGSearch	Return a list of the instances of a specific object type that meet the selection criteria	208
FLGSearchAll	Return a list of the instances of any object type that meet the selection criteria	217

List object types and instances

You can retrieve a list of object types or instances according to their category or relationships.

What you can do with the Information Catalog Manager API calls

API call	Purpose	See:
FLGFoundIn	Return a list of: objects in which a specific instance is contained; objects for which a specific instance is a contact; objects to which a specific instance is attached as a comment; object types for which a specified Programs instance is associated	120
FLGListAnchors	Return a list of the Grouping objects that are not contained by other objects; these top-level Grouping objects are referred to as *anchors*.	149
FLGListAssociates	Return a list of the objects that are: contained by a specified Grouping object; contacts for a specified object; comments attached to a specified object; linked with a specified object; or Programs associated with a specified object type	152
FLGListContacts	Return a list of all Contact objects associated with a specified Grouping or Elemental object	161
FLGListObjTypes	Return a list of all object types	164
FLGListOrphans	Return a list of currently unassociated Attachment, Contact, or Program object instances	167
FLGListPrograms	Return a list of all Programs objects associated with a non-Programs object type	173
FLGNavigate	Return a list of the Grouping or Elemental objects that the specified Grouping object contains	198
FLGWhereUsed	Return a list of the Grouping objects that contain the specified object	238

Copy metadata objects to or from the Information Catalog Manager

You can import or export metadata to or from the information catalog database.

API call	Purpose	See:
FLGExport	Copy and translate the Information Catalog Manager metadata objects to a file in tag language format	113
FLGImport	Interpret and copy metadata objects from a file in tag language format into the information catalog	138
FLGMdisExport	Copy and translate the Information Catalog Manager metadata objects to a file in MDIS-conforming tag language format	113
FLGMdisImport	Interpret and copy metadata objects from an MDIS-conforming tag language file into the information catalog	138

Start external programs

You can start a DOS or Microsoft Windows application from the Information Catalog Manager.

API call	Purpose	See:
FLGOpen	Start an external program using information from the specified object.	202

Confirm or remove changes to the Information Catalog Manager database

You can commit or roll back changes to the Information Catalog Manager database.

API call	Purpose	See:
FLGCommit	Confirm that you want changes to the information catalog database made permanent	74
FLGRollback	Remove changes made to the information catalog database back to the point where changes were last committed.	207

Manage your enterprise information catalogs

You can manage the list of users authorized to perform object management tasks, choose the comment status choices available to users, and propagate deletions from one information catalog to shadow information catalogs in your enterprise.

What you can do with the Information Catalog Manager API calls

API call	Purpose	See:
FLGManageUsers	Update administrators and grant object management authority to specific users	187
FLGManageCommentStatus	Set and update a list of available status choices for users to assign comments	176
FLGManageFlags	Start or stop recording of information catalog deletions (delete history), or retrieve current setting	180
FLGManageTagBuf	Query or reset currently recorded delete history	185
FLGXferTagBuf	Transfer delete history to a tag file for import into other catalogs	241

Issuing an Information Catalog Manager API call

The standard structure for all the Information Catalog Manager API calls is:

```
rc = FLGxxx (parameter,
             parameter,
             parameter,
             .
             .
             &ExtCode);
```

These parameters are typically assigned values or addresses in the code preceding the API call.

`rc` is the variable for the reason code returned by the API call; a reason code of zero (0) means that the API call completed without errors or warnings. `&ExtCode` is the address for the extended code sometimes returned by the API call.

Passing data to and from the Information Catalog Manager API calls

Information catalog API calls receive input and provide output using two mechanisms: parameters and input structures and output structures.

Passing single input values and pointers as parameters

You can use parameters to provide single input values and pointers to output values and data structures.

All API call parameters that are character strings must be passed as strings terminated by a null character, or *null-terminated strings*. Under the **Syntax**

section for each API call in "Chapter 5. The Information Catalog Manager API call syntax" on page 67, the descriptions for such parameters specify the maximum length of the actual data without the null terminator. For example, the length of an object type identifier, **ObjTypeID**, is specified as 6, not 7.

However, the C declarations for such parameters in the examples include the extra byte for the null terminator. For example, if you use the #define constants in the DG2API.H file, a possible declaration for the **ObjTypeID** parameter is:

```
uchar objtypid[FLG_OBJTYPID_LEN+1]
```

(See "Appendix B. The Information Catalog Manager API header file—DG2APIH" on page 245 for a list of the constants in the DG2API.H file.)

Passing multiple values using input structures and output structures

To provide multiple values of input and receive multiple values of output from the Information Catalog Manager API calls, you need to use input structures and output structures.

Input structures and output structures are self-defining data structures; each structure defines the format and meaning of the data that it is passing.

Each self-defining structure must be a contiguous area of storage. Input structures and output structures contain only character data, and cannot contain nulls.

Each input structure and output structure must contain these two areas:

Header area Identifies and defines the size of the structure

Definition area
Defines object area properties

Structures that define or receive values for the properties defined in the definition area must also contain an *object area*, which specifies values for the properties defined in the definition area. Figure 3 on page 22 shows how these three areas are put together.

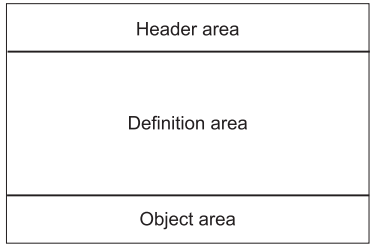

Figure 3. An input or output structure

To pass an input structure to an API call, build the input structure and pass a pointer to the beginning of the input structure as an input parameter for the API call.

To retrieve information from an output structure, pass the address of a null pointer as an input parameter so that the Information Catalog Manager can assign that pointer the address of the beginning of the output structure.

For example, when you pass the API call a pointer named ppListStruct, which contains the address of a null pointer named pOutStruct, the API call then assigns pOutStruct the address of the output structure, as shown in Figure 4.

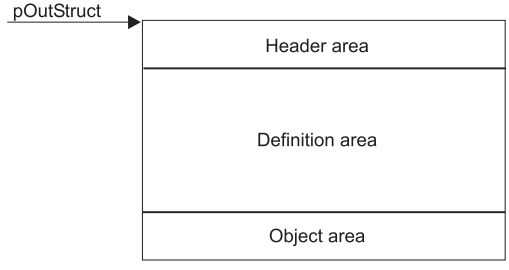

Figure 4. A pointer to an output structure

To avoid running out of memory after several API calls, your program can deallocate the memory allocated for this output structure using the Information Catalog Manager API call FLGFreeMem. For more information about using FLGFreeMem, see "FLGFreeMem" on page 125.

Including header files

The Information Catalog Manager provides C language header files that define the function prototypes of the Information Catalog Manager API calls, constants, data types, and constants for Information Catalog Manager reason codes.

To work with the Information Catalog Manager, your programs must include these header files:

DG2API.H Defines the constants for frequently used values, the Information Catalog Manager-specific data types, and the function prototypes for API calls.

"Appendix B. The Information Catalog Manager API header file—DG2APIH" on page 245 contains a complete list of what is defined in the DG2API.H file.

DG2ERR.H Defines constants for the Information Catalog Manager reason codes.

Your program must contain the following `#define` and `#include` statements to work with the Information Catalog Manager:

```
#define DGWIN32
#include WINDOWS.H
#include DG2API.H
#include DG2ERR.H
```

WINDOWS.H is part of the Microsoft Visual C++ Compiler. This file embeds header files that define standard declarations for Windows data types that are used by the Information Catalog Manager.

An overview of writing a C language program

This section outlines the steps for writing and running a C language program that uses the Information Catalog Manager API calls. Most of this information is standard for any C language program you write.

Creating C language source code

To build an Information Catalog Manager application using C language:

1. Create the source code.
2. Compile the source code using a C compiler.
3. Link the object files with the Information Catalog Manager and C language libraries to produce an executable program.

   The Information Catalog Manager library is DGWAPI.LIB.
4. Execute the application.

Setting up your environment

Use the following steps to set up your environment to compile and run the Information Catalog Manager programs written in the C language:

1. Install the compilers.
2. Verify the LIBPATH.

The LIBPATH= environment variable must include the *x*:**\SQLLIB** directory, where *x* is the drive where you installed DB2 UDB.

3. Set environment variables. You set environment variables either in your AUTOEXEC.BAT or from the Microsoft Visual C++ Compiler menu bar (include file path and library file path).

   The SET INCLUDE= statement must include the *x*:**\SQLLIB\LIB** directory. The directory containing the WINDOWS.H should also be specified on SET INCLUDE=.

   SET LIB= must include the *x*:**\SQLLIB\LIB** directory.

Compiling and linking your application

To compile your application using Microsoft Visual C++ Compiler you need to issue a command such as:

cl /c *filename*.**c**

You might need or want to add other options, depending on the compiler you use and the way you write your program.

To link your program, issue a command such as:

link /dll dgwapi.lib *filename*.**obj**

How to use the Information Catalog Manager API calls in your program

You must follow certain rules and guidelines when you write C language programs that contain the Information Catalog Manager API calls. These guidelines are explained in this section.

Starting your program with FLGInit

When you write a program that issues the Information Catalog Manager API calls, you must issue an FLGInit call before you can issue any other Information Catalog Manager API calls.

FLGInit initializes the Information Catalog Manager, returns the names of properties required for the Information Catalog Manager object types and registrations, and returns environmental information.

Save the information returned by FLGInit. You might need this information for other Information Catalog Manager API calls. If you are using a national language version of the Information Catalog Manager, FLGInit returns the translated names of the properties required for the Information Catalog Manager object types and registrations. You need to use these translated names in the definition area of your input structure when you create or maintain object types and registrations.

See "FLGInit" on page 142 for information about the contents of the FLGInit output structure.

Ending your program with FLGTerm

Your program must issue an FLGTerm call after it finishes using the Information Catalog Manager functions. FLGTerm ends the Information Catalog Manager session and releases resources used by the Information Catalog Manager. See "FLGTerm" on page 223 for more information about this API call.

Protecting your information catalog database when errors occur

Certain Information Catalog Manager errors indicate that some of the metadata in the database on which your information catalog resides might be inconsistent. Therefore, you should write your program to roll back the information catalog database when your program encounters errors. By issuing FLGCommit calls when your API calls succeed and FLGRollback calls when they fail, you protect your information catalog database from becoming inconsistent.

Attention: When your information catalog database is on DB2 you must issue an FLGRollback call if you encounter an error. Otherwise, your information catalog database may be damaged when your program issues FLGTerm.

Setting up Programs objects to start programs

To start a program that works with your data from an Information Catalog Manager application, create a Programs object instance that is associated with the object type that represents that kind of data.

You must define values for three properties in the Programs object instance that identify the program and associate the Programs object instance with an object type, as shown in Table 5.

Table 5. Properties of a Programs object instance that start the program

Property name	Property short name	Value
Start by invoking	STARTCMD	Path and file name of the program to be started, as well as the start options.
Object type this program handles	HANDLES	8-character short name of the object type
Parameter list is	PARMLIST	List of properties in the associated object type the values of which you want to pass to the program as command-line parameters. Each property is delimited by two percent signs (%%)

The value of the Start by invoking (STARTCMD) property has different recommended formats, depending on the program's interface type. Enter the file name of the program and the recommended starting parameters. For Windows NT®, Windows 95, and Windows 98, the recommended starting parameter is `START filename.exe`. The PATH statement must contain the directory where the program is located.

If the file name of the program is in high-performance file system (HPFS) format and contains blanks, then you must surround the path and file name of the program with double quotes, as shown below:

```
""D:\PROGPATH\My Program.EXE""
```

If your program name contains blanks, then you cannot specify any other start options in the STARTCMD property value.

To start a program, issue an FLGOpen call with the Programs object FLGID and object instance FLGID as parameters. For more information about the FLGOpen call, see "FLGOpen" on page 202.

Creating metadata using API calls

The registration, object type, object instances, and relationships build upon one another; therefore, you can only create a set of these entities in a certain order. When creating new object types, object instances, and relationships, you must issue the Information Catalog Manager API calls in the following order:

1. FLGCreateReg
2. FLGCreateType
3. FLGCreateInst
4. FLGRelation

Deleting metadata using API calls

You can, however, delete registration, object type, object instances, and relationships in two manners: conservative (this method is slower), or potentially destructive (yet quicker).

When deleting object types and object instances in a conservative manner, issue the following the Information Catalog Manager API calls for related object instances and object types in the following order:

1. FLGRelation

   You must delete all relationships where the particular object instances are *containers* of other objects before you can delete these object instances. FLGDeleteInst automatically deletes relationships where object instances are *contained* or have associated Contact, Attachment, or linked objects.

2. FLGDeleteInst

   You must delete all object instances of a particular object type before you can delete the object type using FLGDeleteType.

3. FLGDeleteType
4. FLGDeleteReg

You can delete object instances and object types more quickly using the following APIs, but if you are not completely certain of your information catalog's contents, the results can be destructive.

1. FLGDeleteTree

   Simultaneously delete a Grouping object instance and, optionally, all object instances it contains as well as all relationships in which the contained object instances participate.
2. FLGDeleteTypeExt

   Simultaneously delete the object type, object type registration, and all instances of the object type. You must delete individual branches containing objects of other object types before you can delete the object type using FLGDeleteTypeExt.

Specifying the information catalog metadata using the Information Catalog Manager data types

The Information Catalog Manager stores the metadata for an object's properties using four data types, which are defined in Table 6.

Your program may need to make some data conversions to ensure that your metadata is in a valid format.

Table 6. Valid data types for information catalog metadata

Data type	How represented	How an omitted value is represented in input and output structures
CHAR	Occupies its defined length. The value is padded on the right with trailing blanks if the value is shorter than its defined length.	Blanks fill up the value's defined length.
TIMESTAMP	Occupies its full length (26) using the following format: yyyy-mm-dd-hh.mm.ss.nnnnnn	Represented by 26 blanks.
LONG VARCHAR	Preceded by an 8-character length field that specifies the actual length of the following value.	Length field is set to zeros that specifies that no value follows. Example: `00000000`
VARCHAR	Preceded by an 8-character length field that specifies the actual length of the following value.	Length field set to zeros that specifies that no value follows. Example: `00000000`

Including header files

With input structures, the Information Catalog Manager automatically removes trailing blanks from variable-length values and adjusts their lengths accordingly before validating and accepting the request. Therefore, if only blanks are specified for a required value, the request is rejected with a reason code indicating that a required value was not specified. When a value is required, but not available, you can use the not-applicable symbol to avoid errors.

National language considerations

Unless otherwise specified, the Information Catalog Manager commands, parameters, required property short names, data type names, indicator values, and option values are not translated for national language versions, and must be entered in English.

Translated required properties

The 80-byte names of required registration properties and object type properties are translated into the national language.

The English names for the required registration properties are:
- EXTERNAL NAME OF OBJ TYPE
- PHYSICAL TYPE NAME
- DP NAME
- CREATOR
- LAST CHANGED BY
- LAST CHANGED DATE AND TIME

The English names for the required object type properties are:
- Object type identifier
- Instance identifier
- Name
- Last Changed Date and Time
- Last Changed By

The translated names are returned in the output structure produced by the FLGInit call.

Specifying values in languages other than English

Most metadata values stored in an information catalog can be stored in any language. This section describes the guidelines for using SBCS characters and DBCS characters in values with the Information Catalog Manager.

Values that use SBCS characters only
- DP NAME (object type short name) values
- Property short names
- PT NAME (physical type name) values

Values that can use SBCS or DBCS characters

- NAME (external name of an object type) values
- Property names, other than those required for object types and registrations
- Property values for user-defined properties
- Values for the following API call parameters:

FLGCreateReg	pszIconFileID
FLGGetReg	pszIconFileID
FLGExport	pszTagFileID, pszLogFileID, pszIcoPath
FLGImport	pszTagFileID, pszLogFileID, pszIcoPath
FLGInit	pszUserID, pszPassword, pszDatabaseName
FLGManageIcons	pszIconFileID
FLGMdisExport	pszTagFileID, pszLogFileID, pszObjTypeName, pszObjectName
FLGMdisImport	pszTagFileID, pszLogFileID
FLGUpdateReg	pszIconFileID
FLGXferTagBuf	pszTagFileID

Introducing DG2SAMP.C

The Information Catalog Manager provides a sample program, DG2SAMP.C, that you can compile, link, and run. DG2SAMP.C is in the DG2LIB\LIB directory on the drive where you installed the Information Catalog Manager.

This book uses parts of DG2SAMP.C to show how to write applications that use the Information Catalog Manager API calls. DG2SAMP.C issues the following calls:

- FLGCommit
- FLGFreeMem
- FLGGetInst
- FLGInit
- FLGListObjTypes
- FLGRollback
- FLGSearch
- FLGTerm
- FLGTrace
- FLGUpdateInst

For instructions for compiling and linking DG2SAMP.C and an example for running the program, see "Appendix A. Sample program DG2SAMP.C" on page 243.

Chapter 4. The Information Catalog Manager input and output structures

The Information Catalog Manager API calls receive input and provide output using parameters and input structures and output structures. The input structures and output structures allow you to provide multiple values of input and receive multiple values of output from the Information Catalog Manager API calls.

Input structures and output structures are self-defining data structures; each structure defines the format and meaning of the data that it passes.

To pass an input structure to an API call, you need to build the input structure and pass a pointer to the beginning of the input structure as an input parameter for the API call. This process is explained in "Creating input structures for an API call" on page 40.

To retrieve information from an output structure, you need to step through the output structure using one or more pointers. This process is explained in "Reading an output structure resulting from an API call" on page 58.

Although the examples in this book are written in C language, you can create and read input and output structures using any programming language.

Common characteristics of the Information Catalog Manager API input and output structures

The Information Catalog Manager input structures and output structures contain three parts, called *areas*, as shown in Figure 5:

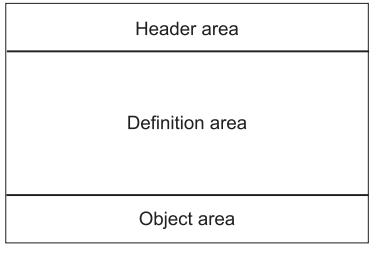

Figure 5. An input or output structure

Header Identifies and defines the size of the structure

Common characteristics of the API input and output structures

Definition Defines object area properties

Object Specifies property values

The entire self-defining structure must be a contiguous area of storage.

Input structures and output structures contain only character data, and cannot contain null characters.

If you omit a value in an input or output structure, use an appropriate number of space characters, called *blanks* in this book, in place of the value to keep the byte offsets of the values consistent with the definition of the input structure and output structure.

The Information Catalog Manager API input structure

Figure 6 shows the general format of the Information Catalog Manager API input structure. The structure consists of three contiguous areas: the header area, the definition area, and the object area. Some Information Catalog Manager API calls require only the first two areas.

The fields of each of the areas are described in the following sections.

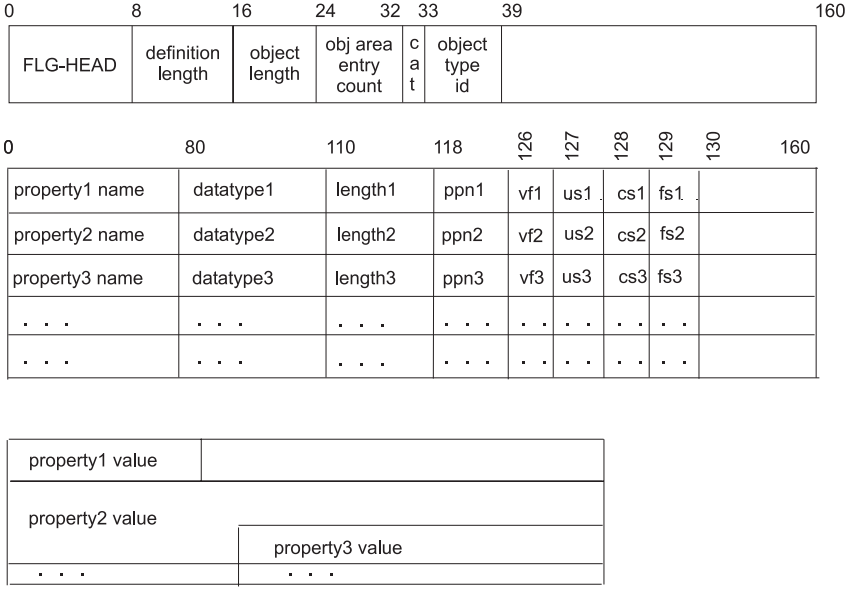

Figure 6. API input structure

The following API calls receive input from an input structure:

FLGAppendType
Adds new properties to an object type

FLGCreateInst
Creates a new object instance

FLGCreateReg
Registers a new object type

FLGCreateType
Creates a new object type

FLGExport Copies and translates the Information Catalog Manager metadata objects to a file in tag language format

FLGManageCommentStatus
Updates the list of available status choices for comments

FLGManageUsers
Updates the administrators and users for an information catalog and identifies extent of each user's authority

FLGSearch Returns a list of the instances of a specific object type that meet the selection criteria

FLGSearchAll Returns a list of the instances of any object type that meet the selection criteria

FLGUpdateInst
Updates information about an object instance

FLGUpdateReg
Updates information about an object type registration

If FLGSearch and FLGSearchAll do not receive an input structure, they attempt to retrieve all objects.

Header area — always required

The header area describes the information in the definition and object areas. Any fields that are not required and are not specified must be set to blanks.

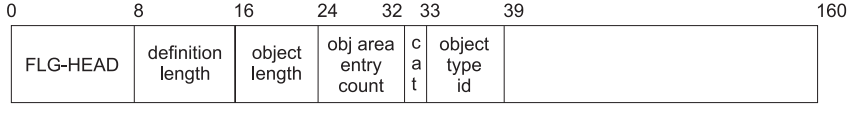

Figure 7. Input structure header area

Table 7 on page 34 describes the meaning of each byte offset position in the header area shown in Figure 7.

Table 7. The input structure header area and its fields

Section from Figure 7 on page 33	Byte offset	Required?	Description
FLG-HEAD	0-7	Always	Structure identifier.
definition length	8-15	Always	Length of the definition area. The value must be a multiple of 160 (160 times the number of definition records).
object length	16-23	Always	Length of the object area. For FLGAppendType and FLGCreateType, this value is zero (00000000).
obj area entry count	24-31	Always	Number of entries (property values) in the object area. The value is the number of properties in the definition area times the number of sets of values described in the object area. For FLGAppendType and FLGCreateType, this value is zero (00000000).
cat	32	Required for: • FLGAppendType • FLGCreateInst • FLGCreateReg • FLGCreateType • FLGUpdateInst • FLGUpdateReg	Category of the object type or object. Valid values are: **G** Grouping **E** Elemental **C** Contact **P** Program **D** Dictionary **S** Support **A** Attachment

Table 7. The input structure header area and its fields (continued)

Section from Figure 7 on page 33	Byte offset	Required?	Description
object type id	33-38	Required for: • FLGAppendType • FLGCreateInst • FLGCreateType • FLGUpdateInst • FLGUpdateReg	System-generated identifier for an object type.
	39-159	Always	Should be left blank.

Definition area — always required

The definition area contains a set of property definitions required as input by a particular the Information Catalog Manager API function.

Table 8 shows what the information in the definition area means for different API calls that use input structures.

Table 8. The meaning of the definition area for different API calls

API calls	Information in the definition area
FLGAppendType FLGCreateInst FLGCreateReg FLGCreateType FLGUpdateInst FLGUpdateReg	Definition of the set of properties that define the object registration, object type, or object instance
FLGSearch FLGSearchAll	Definition of the set of properties that describe the selection criteria
FLGExport	Definition of the properties that specify the metadata to be exported
FLGManageCommentStatus	Definition of the set of properties that specify Comments status choices
FLGManageUsers	Definition of the set of properties that describe the Information Catalog Manager users

Each property in the definition area is defined by a set of formatted specifications. Table 9 on page 36 describes the byte offset positions shown in Figure 8 on page 36.

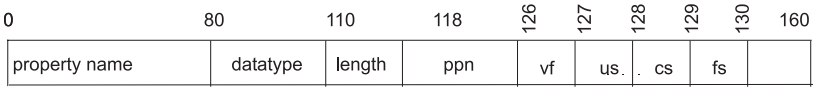

Figure 8. Input structure definition record

Table 9. The input structure definition area and its fields

Section from Figure 8	Byte offset	Required?	Description
property name	0-79	Always	External name of the property.
datatype	80-109	Always	The data type of the property. Valid values are: **CHAR** Fixed-length character data. Maximum length is 254. **VARCHAR** Variable-length character data. Maximum length is 4000. **LONG VARCHAR** Variable-length character data. Maximum length is 32700. **TIMESTAMP** Time stamp in the form of: `yyyy-mm-dd-hh.mm.ss.nnnnnn` Timestamp length is 26.
length	110-117	Always	Maximum length of the property value.

Table 9. The input structure definition area and its fields (continued)

Section from Figure 8 on page 36	Byte offset	Required?	Description
ppn	118-125	Required for: • FLGAppendType • FLGCreateInst • FLGCreateReg • FLGCreateType • FLGManage-CommentStatus • FLGSearch • FLGSearchAll • FLGUpdateInst • FLGUpdateReg For other API calls this field is unused and left blank	Property short name
vf	126	Required for: • FLGAppendType • FLGCreateInst • FLGCreateReg • FLGCreateType • FLGUpdateInst • FLGUpdateReg For other API calls this field is unused and left blank.	Value flag specifying whether a property is required, optional, or system-generated. Valid values are: **R** Required **O** Optional **S** System-generated

Table 9. The input structure definition area and its fields (continued)

Section from Figure 8 on page 36	Byte offset	Required?	Description
us	127	Required for the following API calls: • FLGCreateInst • FLGCreateType • FLGUpdateInst For other API calls, this field is unused and left blank.	Universal unique identifier (UUI) sequence number, which specifies that a property is part of the UUI. Valid values are: **1** UUI Part 1 **2** UUI Part 2 **3** UUI Part 3 **4** UUI Part 4 **5** UUI Part 5 **(blank)** Not part of the UUI At least one property must be specified as UUI Part 1 for any object type. See the *Information Catalog Manager Administration Guide* for more information about defining UUI parts.
cs	128	Required for the following API calls: • FLGSearch • FLGSearchAll For other API calls, this field is unused and left blank.	Case-sensitivity flag. Valid values are: **Y** Case-sensitive **N** Not case-sensitive See "FLGSearch" on page 208 and "FLGSearchAll" on page 217 for information about using the case-sensitivity flag.
fs	129	Required for the following API calls: • FLGSearch • FLGSearchAll For other API calls, this field is unused and left blank.	Fuzzy search flag. Valid values are: **Y** Fuzzy search **N** Not a fuzzy search See "FLGSearch" on page 208 and "FLGSearchAll" on page 217 for information about using the fuzzy search flag.
	130-159	Always	Reserved section. Should be left blank.

Object area — Required when defining values

The object area contains the values for the properties defined in the definition area. The values must appear in the order defined in the definition area.

The object area for an input structure contains only one value per property defined in the definition area for all APIs except FLGExport and FLGManageUsers. For FLGExport and FLGManageUsers, the object area can contain more than one value per property defined in the definition area.

The object area is required for the following API calls:
- FLGCreateInst
- FLGCreateReg
- FLGExport
- FLGManageCommentStatus
- FLGManageUsers
- FLGSearch
- FLGSearchAll
- FLGUpdateInst
- FLGUpdateReg

You can determine how to represent each value using the following rules:

Data type	How to represent the value in the object area
VARCHAR	Value is preceded by an 8-character length field that specifies the actual length of the value. Trailing blanks are automatically removed from these values; the Information Catalog Manager adjusts the length field accordingly.
LONG VARCHAR	Value is preceded by an 8-character length field that specifies the actual length of the value. Trailing blanks are automatically removed from these values; the Information Catalog Manager adjusts the length field accordingly.
CHAR	Value occupies the number of bytes defined by the property's length field in the definition area and is padded on the right with blanks to fill the defined length.
TIMESTAMP	26 bytes

Creating input structures for an API call

Follow these steps to create an input structure:

1. Define lengths and values using DG2API.H
2. Calculate the size of the entire output structure
3. Define the header area
4. Define the definition area
5. Define the object area

Defining lengths and values using DG2API.H

The Information Catalog Manager provides a C language header file named DG2API.H that defines many of the value lengths and valid values that you need to create input structures and read output structures. You can include (using the `#include` statement) this file in your program so that you do not need to write the code for certain data types, structures, and function prototypes yourself.

DG2API.H contains type definition (`typedef`) declarations of the structures needed to build the header and definition areas, as shown in Figure 9 on page 41. (In Figure 9 on page 41, `WINDOWS` refers only to Microsoft Windows 3.1.)

```
#pragma pack(1)

/* Structure definition for the FLG header area */
  typedef   struct _FLG_HEADER_AREA {
     UCHAR        pchHIdent         [ FLG_H_IDENT_LEN      ];
     UCHAR        pchHDefLength     [ FLG_H_DEFAREA_LEN    ];
     UCHAR        pchHObjLength     [ FLG_H_OBJAREA_LEN    ];
     UCHAR        pchHObjEntryCount [ FLG_H_OBJAREAENT_LEN ];
     UCHAR        pchHCategory      [ FLG_H_CATEGORY_LEN   ];
     UCHAR        pchHObjTypeId     [ FLG_H_OBJTYPID_LEN   ];
     UCHAR        pchHReserved      [ FLG_H_RESERVED_LEN   ];
  } FLGHEADERAREA;
#ifdef WINDOWS
  typedef FLGHEADERAREA __huge *PFLGHEADERAREA;
#else
  typedef FLGHEADERAREA *PFLGHEADERAREA;
#endif

/* Structure definition for the FLG definition area */
  typedef   struct _FLG_DEFINITION_AREA {
     UCHAR        pchDPropName      [ FLG_D_PROPNM_LEN   ];
     UCHAR        pchDDataType      [ FLG_D_DATATYP_LEN  ];
     UCHAR        pchDDataLength    [ FLG_D_DATA_LEN     ];
     UCHAR        pchDTagName       [ FLG_D_PPN_LEN      ];
     UCHAR        pchDVF            [ FLG_D_VF_LEN       ];
     UCHAR        pchDUS            [ FLG_D_US_LEN       ];
     UCHAR        pchDCS            [ FLG_D_CS_LEN       ];
     UCHAR        pchDFS            [ FLG_D_FS_LEN       ];
     UCHAR        pchDReserved      [ FLG_D_RESERVED_LEN ];
  } FLGDEFINITIONAREA;
#ifdef WINDOWS
  typedef FLGDEFINITIONAREA __huge *PFLGDEFINITIONAREA;
#else
  typedef FLGDEFINITIONAREA *PFLGDEFINITIONAREA;
#endif
```

Figure 9. DG2API.H: Structure definitions for the header and definition areas

Variables starting with FLG_D or FLG_H are lengths for the structure parts that are defined in DG2API.H.

See "Appendix B. The Information Catalog Manager API header file—DG2APIH" on page 245 for a list of all the constants defined in the DG2API.H file.

You can use these defined structures to define the storage required for the header and definition areas of the input structure. Figure 10 on page 42 shows a part of DG2SAMP.C that uses data types defined in the DG2API.H header file to define the structures later used to store the header and definition areas of an input structure.

```
//  This structure defines the input structure for FLGSearch.
typedef _Packed struct SEARCH_STRUCT  {
   FLGHEADERAREA        srchHdr;
   FLGDEFINITIONAREA    srchDef;
   OBJECTAREA           Item;
} SEARCHSTRUCT;
typedef SEARCHSTRUCT  *PSEARCHSTRUCT;
```

Figure 10. DG2SAMP.C: Defining the header and definition areas

To ensure that the input structure is defined as contiguous storage, Figure 9 on page 41 uses a `#pragma pack(1)` instruction, and Figure 10 uses a `typedef _Packed struct` definition. If you build input structures using another programming language, be aware that you might need to issue similar commands to define the input structure as contiguous storage.

Calculating the size of the entire input structure

You need to calculate the size of the entire input structure so that you can allocate the amount of storage for the input structure. To make this calculation, you need to know the following values:

- Number of properties defined in the definition area

  This value depends on the number of properties required by the API call. You use this value to calculate the length of the definition area.
- Lengths of the values in the object area. You add these values together to get the length of the object area.

DG2API.H provides variables that define the length of the header area (FLG_HEADER_SIZE) and the length of a single definition record (FLG_DEFINITION_SIZE).

Calculating the definition area length

To calculate the definition area length, multiply the fixed length of each definition record (160) by the number of records needed to define your data, as shown in Figure 11.

```
Definition_area_length = number_of_properties × FLG_DEFINITION_SIZE
```

Figure 11. Calculating the definition area length

DG2API.H provides the variable FLG_DEFINITION_SIZE, defined as 160, to help you define this calculation in your code.

You will need this value to define the definition area length field of the header area, as shown in “Defining the header area” on page 44.

Calculating the object area length

The object area length is the sum of the lengths of all the values that go into the object area.

You will need this value to define the object area length field of the header area, as shown in "Defining the header area" on page 44.

If you are creating an input structure for an API call that does not require or expect an object area, the value in the object area is zero (00000000).

To calculate the exact object area length, you need the length of all of the values in your object area. For CHAR and TIMESTAMP values, use the length defined in the definition area. However, for LONG VARCHAR and VARCHAR values, you need to check the length for each value and include the 8-byte length field as part of the length value. The formula for this calculation is shown in Figure 12.

```
Object_area_length = length_of_property1 +
                     length_of_property2 +
                     length_of_property3 +
                     .
                     .
                     .
```

Figure 12. Calculating the exact object area length

You can also define your object area to contain the longest possible value for all properties, including VARCHAR and LONG VARCHAR values. With this method, you can add the maximum data lengths for all the properties together to ensure that the values you define for the object area will fit in the allocated storage. For VARCHAR and LONG VARCHAR properties, be sure to include the 8-byte length field as part of the maximum length value. The formula for this calculation is the following:

```
Length_of_object_area = maximum_length_of_property1 +
                        maximum_length_of_property2 +
                        maximum_length_of_property3 +
                        .
                        .
                        .
```

Figure 13. Calculating the maximum possible object area length

Be aware, however, that this method can waste a lot of storage, especially if several of your properties are LONG VARCHAR fields with a maximum length of 32700 bytes.

Adding all the parts together

The entire formula for determining the storage you need to allocate is shown in Figure 14.

```
Structure_size = FLG_HEADER_SIZE +
                 Definition_area_length +
                 Object_area_length
```

Figure 14. Calculating the required storage for an input structure

Defining the header area

Because the input structure is a self-defining structure, there are several values in the header area that define the structure's size and format. To define these values properly, you need to consider the entire set of information and the structure you need to create.

The header area is 160 bytes. Each byte position must be assigned a value; if you do not specify a value, you must define a blank for that position. One way of defining one or more byte positions as blanks is to use the C language `memset` function to set the entire structure to FLG_BLANK or all zero characters first, and then to use the C language `memcpy` function to copy only the information that needs to be set to something else. This method also makes it easier to use the constants defined in DG2API.H, because you only need to worry about overlaying blanks or zeroes, not about padding the values to match the data length.

Complete specifications for each byte of the header area are discussed in "Header area — always required" on page 33.

The syntax for the header area for each API call is discussed in "Chapter 5. The Information Catalog Manager API call syntax" on page 67.

Although some values in the header area not required for certain API calls, you need to define the header area to contain the byte offset positions shown in Figure 15.

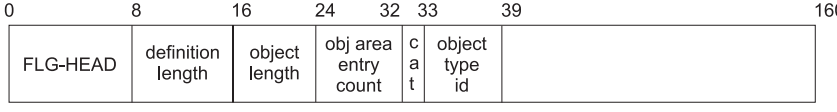

Figure 15. The header area

These byte offset positions are described in Table 7 on page 34. Table 10 on page 45 lists constants in DG2API.H that can help you define the header area.

Table 10. Header area byte offset positions and useful constants defined in DG2API.H

Bytes	Contents	Useful constants defined in DG2API.H	Value
0-7	FLG-HEAD	FLG_H_IDENT	FLG-HEAD
8-15	Definition area length	FLG_DEFINITION_SIZE	160; length of one definition area record
16-23	Object area length		
24-31	Object area entry count		
32	Category	FLG_GROUPING_OBJ FLG_ELEMENTAL_OBJ FLG_CONTACT_OBJ FLG_DICTIONARY_OBJ FLG_PROGRAM_OBJ FLG_SUPPORT_OBJ FLG_ATTACHMENT_OBJ	G E C D P S A
33-38	Object type ID		
39-159	Reserved area (always blank)		

When you define the header area, three values depend on the content of the definition and object areas:

- Definition area length (bytes 8-15)

  You probably already calculated this value to allocate storage for the input structure. To review the description of this calculation, see "Calculating the definition area length" on page 42.

- Object area length (bytes 16-23)

  You probably already calculated this value to allocate storage for the input structure. To review the description of this calculation, see "Calculating the object area length" on page 43.

- Object area entry count (bytes 23-31)

  For all API calls requiring an input structure except FLGExport and FLGManageUsers, the object area entry count equals the number of properties in the definition area. For FLGExport, the object area entry count equals five times the number of objects specified to be exported. For FLGManageUsers, the object area entry count equals two for each user added or updated.

Defining the definition area

To define the definition area, you need to know what information the API call requires in the input structure.

Each record of the definition area is 160 bytes long. Each byte position must be assigned a value; even if you do not specify a value, you must define a blank for that position. One way of defining one or more byte positions to blanks is to use the C language `memset` function to set the entire structure to FLG_BLANK first, and then to use the C language `memcpy` function to copy only the information that needs to be set to something else. This method also makes it easier to use the constants defined in DG2API.H, because you only need to worry about overlaying blanks, not about padding the values to match the data length. Although some of the values are not required for certain API calls, the definition area must always contain the full 160 bytes as shown in Figure 16.

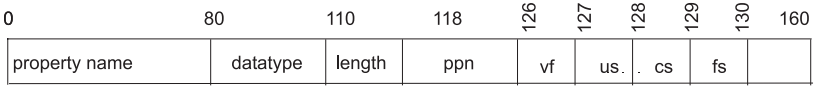

Figure 16. A record in the definition area

These byte offset positions are described in Table 9 on page 36. Table 11 lists constants in DG2API.H that can help you define the definition area.

Table 11. Definition area byte offset positions and useful constants defined in DG2API.H.

Bytes	Contents	Useful variables in DG2API.H	Values
0-79	Property name		
80-109	Data type	FLG_DTYPE_CHAR FLG_DTYPE_VARCHAR FLG_DTYPE_LONGVARCHAR FLG_DTYPE_TIMESTAMP	CHAR VARCHAR LONG VARCHAR TIMESTAMP
110-117	Data length		
118-125	Property short name	FLG_PPN_OBJTYPID FLG_PPN_INSTIDNT FLG_PPN_INST_NAME FLG_PPN_UPDATIME FLG_PPN_UPDATEBY FLG_PPN_EXTERNAL_NAME FLG_PPN_PTNAME FLG_PPN_DPNAME FLG_PPN_CREATOR	OBJTYPID INSTIDNT NAME UPDATIME UPDATEBY NAME PTNAME DPNAME CREATOR
126	Value flag	FLG_REQUIRED FLG_OPTIONAL FLG_SYSTEM	R O S

Table 11. Definition area byte offset positions and useful constants defined in DG2API.H. (continued)

Bytes	Contents	Useful variables in DG2API.H	Values
127	UUI sequence number	FLG_UUI_1 FLG_UUI_2 FLG_UUI_3 FLG_UUI_4 FLG_UUI_5 FLG_BLANK	1 2 3 4 5
128	Case-sensitivity flag	FLG_YES FLG_NO	Y N
129	Fuzzy search flag	FLG_YES FLG_NO	Y N
130-159	Reserved area (always blank)		

For more information about the specific meanings for all the byte positions in the definition area, see "Definition area — always required" on page 35. For more information about the definition for the API call you are using, see "Chapter 5. The Information Catalog Manager API call syntax" on page 67.

Defining the object area

How you define the values in your object area depends on the data type of each property being defined. CHAR and TIMESTAMP values are relatively straightforward because they have fixed lengths, but variable values (VARCHAR and LONG VARCHAR) are more complicated.

TIMESTAMP values have a fixed length and format.

CHAR values are left-justified and padded with trailing blanks to fill the defined length, as in this example:

```
'My example                    '
```

All values must be character data. If the value is numeric, you must convert it to character data.

Null characters are not permitted in any value. If the value you specify does not fill the entire fixed length, you must define blanks or zeroes for the unfilled positions. One way of defining blanks or zeroes for unused byte positions is to use the C language `memset` function to set the entire structure to FLG_BLANK or zero characters ('0' or 0x30) first, and then to use the C language `memcpy` function to copy only the information that needs to be set to

something else. This method also makes it easier to use the constants defined in DG2API.H, because you only need to worry about overlaying blanks, not about padding the values to match the data length.

To specify VARCHAR and LONG VARCHAR values, include an extra 8 bytes before the value to specify the length of the value. For example, the value you need to specify for a VARCHAR value of "`Employee records -- Southwest Region`" would be

```
00000036Employee records -- Southwest Region
```

Because this is a VARCHAR value, you do not need to pad the value with trailing blanks.

Example of defining header, definition, and object areas

This section discusses the parts of DG2SAMP.C that define an input structure.

Calculating the object area length

The code shown in Figure 17 calculates the object area length for an input structure.

```
//-------------------------------------------
// Build input structure for FLGSearch
//-------------------------------------------
printf ("Enter object instance name:\n");
gets(pszObjInstName);                          1
ulInstValLen = strlen(pszObjInstName);         2
ulInstLen = (FLG_VARIABLE_DATA_LENGTH_LEN + ulInstValLen);   3
convertultoa(ulInstLen, pszLength);            4
```

Figure 17. DG2SAMP.C: Determining the object area length

The code in Figure 17 performs the following steps for determining the object area:

1. Sets pszObjInstName to the object instance name entered by the user.
2. Determines the length of the object instance name
3. Adds the length of the variable data length field (8) to the length of the object instance name
4. Converts the object area length value to character data

Defining the header area

The code in Figure 18 on page 49 shows how DG2SAMP.C defines the header area of the input structure for FLGSearch. This header area contains the same values as shown in Figure 19 on page 49.

```
//-------------------------------------------
//  Header
//-------------------------------------------
memset(&(SearchStruct.srchHdr), FLG_BLANK, FLG_HEADER_SIZE);   1

memcpy(&SearchStruct.srchHdr.pchHIdent, FLG_H_IDENT, FLG_H_IDENT_LEN);                  2
memcpy(&SearchStruct.srchHdr.pchHDefLength, "00000160", FLG_H_DEFAREA_LEN);             3
memcpy(&SearchStruct.srchHdr.pchHObjLength, pszLength , FLG_H_OBJAREA_LEN);             4
memcpy(&SearchStruct.srchHdr.pchHObjEntryCount, "00000001", FLG_H_OBJAREAENT_LEN);      5
```

Figure 18. DG2SAMP.C: Defining the header area

The code in Figure 18 performs the following steps for defining a header area using C language.

1 Sets the entire header area to blanks.

2 Sets bytes 0-7 to the identifier (FLG_HEAD).

3 Sets the definition length to 160.

4 Sets the object area length. This length was calculated earlier in the program.

5 Sets the object area entry count to 1.

Figure 19 shows the storage defined by the C language code in Figure 18.

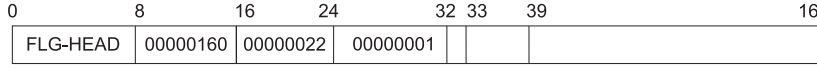

Figure 19. Defined header area—SearchStruct.srchHdr

Defining the definition area

The code in Figure 20 on page 50 shows how DG2SAMP.C defines the definition area of the input structure for FLGSearch. This definition area contains the values shown in Figure 21 on page 50.

```
//-------------------------------------------
// Definition area
//-------------------------------------------
memset(&(SearchStruct.srchDef), FLG_BLANK, FLG_DEFINITION_SIZE);      1
memcpy(&SearchStruct.srchDef.pchDPropName,
"Name                                                                ",FLG_D_PROPNM
memcpy(&SearchStruct.srchDef.pchDDataType, "VARCHAR                ", FLG_D_DATATYP_LEN)
memcpy(&SearchStruct.srchDef.pchDDataLength, "00000080", FLG_D_DATA_LEN);   4
memcpy(&SearchStruct.srchDef.pchDTagName, "NAME    ", FLG_D_PPN_LEN);       5
memset(SearchStruct.srchDef.pchDCS, 'N', FLG_D_CS_LEN);   6
memset(SearchStruct.srchDef.pchDFS, 'N', FLG_D_FS_LEN);   7
```

Figure 20. DG2SAMP.C: Defining the definition area

The code in Figure 20 performs the following steps for defining a record in the definition area using the C language:

1. Sets the entire definition record to blanks
2. Sets the property name to Name
3. Sets the data type to VARCHAR
4. Sets the data length to 80
5. Sets the property short name to NAME
6. Sets the case-sensitivity flag to N
7. Sets the fuzzy search flag to N

Figure 21 shows the storage defined by the C language code in Figure 20.

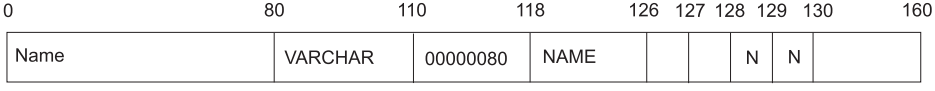

Figure 21. Defined definition area—SearchStruct.srchDef

Defining the object area

Figure 22 on page 51 shows how DG2SAMP.C defines the object area of the input structure for FLGSearch. This object area contains values shown in Figure 23 on page 51.

```
//-------------------------------------------
// Object area
//-------------------------------------------
memset(&(SearchStruct.Item), FLG_BLANK, FLG_INST_NAME_LEN + FLG_VARIABLE_DATA_LENGTH_LEI
convertultoa(ulInstValLen, pszNameLength);                                    2
pszInstanceName=strncat(pszNameLength,pszObjInstName,ulInstValLen);             3
memcpy(&SearchStruct.Item.Name, pszInstanceName, ulInstLen);                  4
```

Figure 22. DG2SAMP.C: Defining the object area

The code in Figure 22 performs the following steps for defining an object area using the C language:

1 Sets the object area to blanks

2 Converts the length of the Name value to character data

3 Concatenates the length of the VARCHAR value with the value

4 Sets the object area to the value length and the value

Figure 23 shows the storage defined by the C language code in Figure 22.

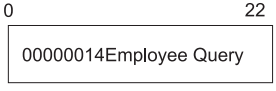

Figure 23. Defined object area—SearchStruct.Item

The Information Catalog Manager API output structure

Figure 24 on page 52 shows the general format of the Information Catalog Manager API output structure. The output structure consists of three contiguous areas: the header area, the definition area, and the object area. Some Information Catalog Manager API calls (for example, FLGGetType) produce only the first two areas.

When your program calls an API call that produces an output structure, it passes a pointer to a null pointer as a parameter. The API call then assigns the address of the output structure to the null pointer.

To avoid running out of memory after several API calls, your program can deallocate the memory allocated for this output structure using the Information Catalog Manager API call FLGFreeMem. For more information about FLGFreeMem, see “FLGFreeMem” on page 125.

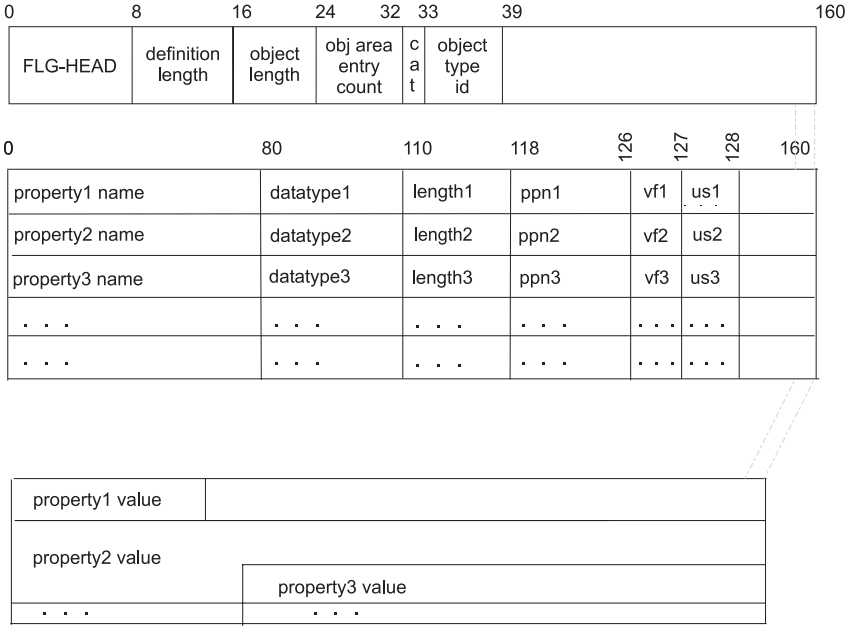

Figure 24. API output structure

The following API calls produce output structures to return data:

FLGDeleteTree
Returns a list of deleted object instances

FLGFoundIn Returns a list of instances or object types in which a specified instance is found

FLGGetInst Gets information about an object instance

FLGGetReg Gets the information for an object type registration

FLGGetType Gets information about an object type

FLGInit Allocates required resources and initializes the Information Catalog Manager client

FLGListAnchors
Returns a list of the instances of the Grouping objects that are not contained by other objects; these top-level Grouping objects are referred to as *anchors*.

FLGListAssociates
Returns a list of the associate instances for a specified instance or object type

FLGListContacts
Returns a list of all Contact object instances for a specified instance

FLGListObjTypes
Returns a list of all object types

FLGListOrphans
Returns a list of instances for a specified object type that are not currently associated with any other instances

FLGListPrograms
Returns a list of all Program objects

FLGManageCommentStatus
Updates the list of available status choices for comments

FLGManageUsers
Updates the administrators and users for an information catalog and identifies extent of each user's authority

FLGNavigate Returns a list of the Grouping or Elemental objects that the specified Grouping object contains

FLGSearch Returns a list of the instances of a specific object type that meet the selection criteria

FLGSearchAll Returns a list of the instances of any object type that meet the selection criteria

FLGWhereUsed
Returns a list of the Grouping objects that contain the specified object

Header area — always present

The header area describes the information in the definition and object areas. The byte-offset positions of the header area are shown in Figure 25 and described in Table 12.

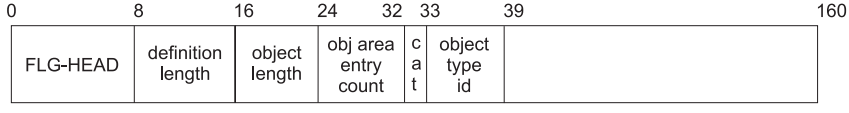

Figure 25. Output structure header area

Table 12. The output structure header area and its fields

Section from Figure 25	Byte offset	Present?	Description
FLG-HEAD	0-7	Always	Structure identifier.

Table 12. The output structure header area and its fields (continued)

Section from Figure 25 on page 53	Byte offset	Present?	Description
definition length	8-15	Always	Length of the definition area. Value is a multiple of 160 (number of properties times the length of each definition record).
object length	16-23	Always	Length of the object area. If no data is returned, then the length of the object area is zero (00000000).
obj area entry count	24-31	Always	Number of individual property values entered in the object area. Value is the number of properties in the definition area times the number of sets of values described in the object area. If no data is returned, then the length of the object area is zero (00000000).
cat	32	Present with: • FLGGetInst • FLGGetReg • FLGGetType	Category of the object type or object. Valid values are: **G** Grouping **E** Elemental **C** Contact **P** Program **D** Dictionary **S** Support **A** Attachment
object type id	33-38	Present with: • FLGGetInst • FLGGetReg • FLGGetType	System-generated identifier for an object type.
	39-159	Always	Should be left blank.

Definition area — always present

The definition area contains a set of property definitions produced as output values by a particular Information Catalog Manager API function.

Table 13 shows the meaning of the definition area for the API calls that produce output structures.

Table 13. The meaning of the definition area for different API calls

API calls	Information in the definition area
FLGGetInst FLGGetReg FLGGetType FLGDeleteTree	Definition of the set of properties that define the object registration, object type, or object instance
FLGInit	Information about the Information Catalog Manager environment
FLGFoundIn FLGListAnchors FLGListAssociates FLGListContacts FLGListObjTypes FLGListOrphans FLGListPrograms FLGManage-CommentStatus FLGManageUsers FLGNavigate FLGSearch FLGSearchAll FLGWhereUsed	Definition of the set of properties that describe each item returned by one of these API calls

Figure 26 shows the byte-offset positions for a record in the definition area.

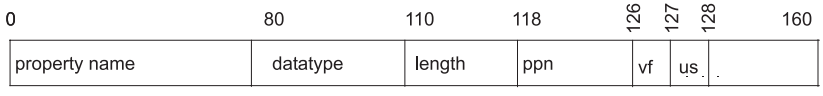

Figure 26. A record in the definition area

Each property in the set is defined by a set of formatted specifications, as described in Table 14 on page 56.

Table 14. The output structure definition area and its fields

Section from Figure 26 on page 55	Byte offset	Present?	Description
property name	0-79	Always	External name
datatype	80-109	Always	The data type of the property. Valid values are: **CHAR** Fixed-length character data. Maximum length is 254. **VARCHAR** Variable-length character data. Maximum length is 4000. **LONG VARCHAR** Variable-length character data. Maximum length is 32700. **TIMESTAMP** Time stamp in the form of: `yyyy-mm-dd-hh.mm.ss.nnnnnn` Timestamp length is 26.
length	110-117	Always	Maximum length of the property value in the object area.
ppn	118-125	Present with: • FLGGetInst • FLGGetReg • FLGGetType • FLGManage-CommentStatus For other API calls, this field is unused and left blank	Property short name

Table 14. The output structure definition area and its fields (continued)

Section from Figure 26 on page 55	Byte offset	Present?	Description
vf	126	Present with: • FLGGetInst • FLGGetReg • FLGGetType For other API calls, this field is unused and left blank	Value flag specifying whether a property is required, optional, or system-generated. Valid values are: **R** Required **O** Optional **S** System-generated
us	127	Present for the following API calls: • FLGGetInst • FLGGetType For other API calls, this field is unused and left blank	Universal Unique Identifier (UUI) sequence number that specifies that a property is part of the UUI. Valid values are: **1** UUI Part 1 **2** UUI Part 2 **3** UUI Part 3 **4** UUI Part 4 **5** UUI Part 5 **(blank)** Not part of the UUI See the *Information Catalog Manager Administration Guide* for more information about UUI parts.
	128-159	Always	Reserved section. Is left blank.

Object area — Present when retrieving information

The object area contains the values for the properties defined in the definition area. The values appear in the order defined in the definition area.

The object area is included in the output structure for the following API calls:

FLGDeleteTree
FLGFoundIn
FLGGetInst
FLGGetReg
FLGInit
FLGListAnchors

FLGListAssociates
FLGListContacts
FLGListObjTypes
FLGListOrphans
FLGListPrograms
FLGManageCommentStatus
FLGManageUsers
FLGNavigate
FLGSearch
FLGSearchAll
FLGWhereUsed

You can determine the size of each value using the following rules:

Data type	Rules for value size
VARCHAR	Value is preceded by an 8-character length field that specifies the actual length of the value.
LONG VARCHAR	Value is preceded by an 8-character length field that specifies the actual length of the value.
CHAR	Value occupies the number of bytes defined by the property's length field in the definition area and is padded on the right with blanks to fill the defined length.
TIMESTAMP	26 bytes.

Reading an output structure resulting from an API call

The Information Catalog Manager API calls that return information put that information into an output structure.

To read an output structure, consider the structure as a whole, because different parts of the structure define the meaning of other parts of the structure.

For API calls that return lists of object instances, the object area can contain more than one value for each property. The object area can contain several sets of values that map to the properties defined in the definition area.

Using pointers to read an output structure

To read values in the output structure, define two or more pointers to the structure, using the pointer value returned by the API call.

Reading an output structure resulting from an API call

When your program issues an API call that produces an output structure, your program must define a pointer that contains the address of a null pointer and pass this defined pointer to the API call as a parameter. The API function then assigns the null pointer the address of the output structure.

You need to define a second pointer that will step through the header and definition areas of the structure, and a third that will step through the object area.

In Figure 27, pOutStruct is the pointer to the beginning of the output structure. You can then define pReadStruct to step through the header area and definition area, and pObjArea to step through the object area.

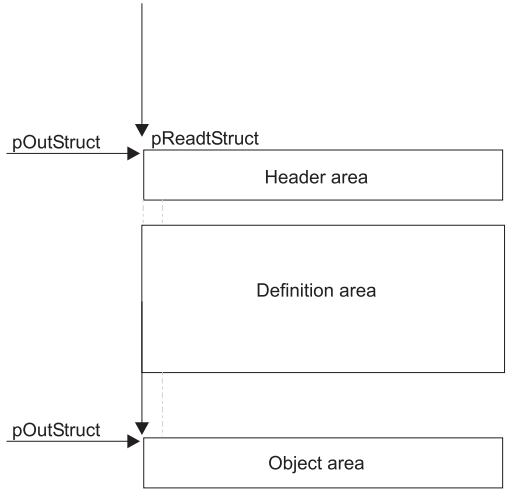

Figure 27. Defining pointers that step through the output structure

Depending on your needs, you can either read the values of the structure in the order they are returned, or you can search for a specific value. In either case, you need to:

1. Calculate the number of properties and the number of objects returned
2. Find the data type and data length for each property
3. Step through the object area to read or locate values

Reading values using DG2API.H

The Information Catalog Manager provides a header file named DG2API.H that defines many of the value lengths and valid values that you need to read output structures. You can use these lengths to write the C language code you need to step through the header, definition, and object areas.

See "Appendix B. The Information Catalog Manager API header file—DG2APIH" on page 245 for a complete list of the constants defined in the DG2API.H file.

Calculating the number of properties in the output structure

Certain API calls return an unknown number of properties, so you need to calculate this number.

Set a pointer to the beginning of the output structure using the pointer address returned by the API call.

To calculate the number of properties in the definition area, divide the numeric value of the definition length area of the header area (bytes 8-15) by the length of an individual record in the definition area (160). You need to convert the definition length character string to an integer value to perform this calculation.

DG2API.H provides the variable FLG_DEFINITION_SIZE to help you write this calculation:

```
number_of_properties = definition_length_integer_value / FLG_DEFINITION_SIZE
```

Figure 28. Calculating the number of properties

Calculating the number of sets of values returned

To calculate the number of sets of values returned in the output structure, divide the object area entry count shown in Figure 29 by the number of properties in the structure, as shown in Figure 30.

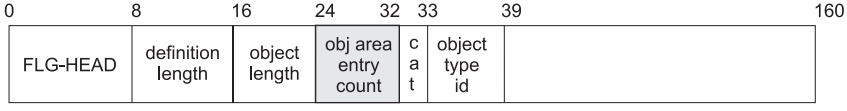

Figure 29. The object area entry count in the header area

```
number_of_sets_of_values = object_area_entry_count / number_of_properties
```

Figure 30. Calculating the number of sets of values

The fields in the header area are in character format and must be converted to numeric format for use in the calculation in Figure 30. You can use the structures defined in DG2API.H to arrive at the calculation in Figure 30.

Reading the property data types and lengths in the definition area

To read the property data types and lengths, define a pointer and perform pointer arithmetic to read the correct values in the definition area. The

location of the data types and lengths of the first property are highlighted in Figure 31.

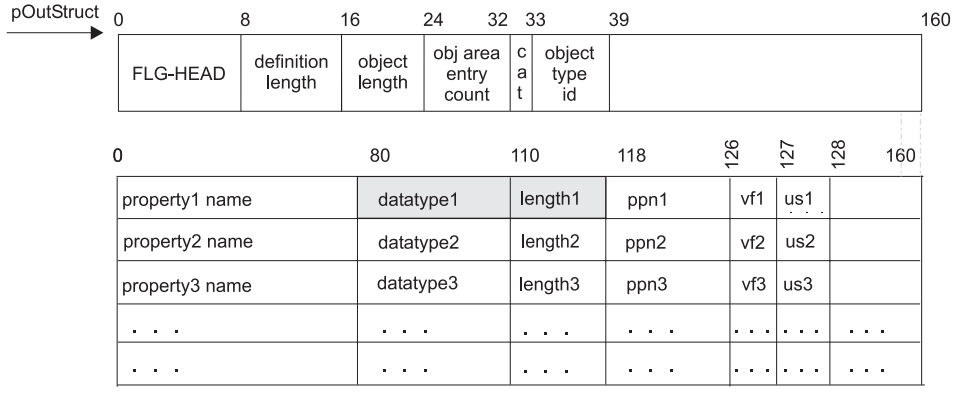

Figure 31. The data type and data length of the first property

To read the data type for the first property in the definition area, add the length of the header area and the property name field of the first definition record to the location of the pointer to the output structure, as shown in Figure 32.

```
pLocationOfDataType = pOutStruct +
                      FLG_HEADER_SIZE +
                      FLG_D_PROPNM_LEN
```

Figure 32. Calculating the position of the data type value

`pOutStruct` is the pointer to the output structure, FLG_HEADER_SIZE is the length of the header area, and FLG_D_PROPNM_LEN is the length of the property name field. You can now save the value at this location in another variable.

To read the data length for the first property in the definition area, add the length of the data type field to the pointer you calculated to get to the data type, as shown in Figure 33.

```
pLocationOfDataLen = pLocationOfDataType +
                     FLG_D_DATATYP_LEN
```

Figure 33. Calculating the position of the data length value

`pLocationOfDataType` is a pointer to the data type field in the definition record and FLG_D_DATATYP_LEN is the length of the data type field.

To read the data types and lengths of other properties, continue to add offset values. To get to the data type field for the next property, you can add the length of an entire data record (160) to the pointer to the data type for the current property as shown in Figure 34.

```
pLocationOfDataType = pLocationOfDataType + FLG_DEFINITION_SIZE
```

Figure 34. Calculating the position of the next data type value

FLG_DEFINITION_SIZE is 160 bytes.

Stepping through the object area to read values

To read a value in the object area, you need to calculate its position using pointer arithmetic. You need to know the data type and length of the properties to calculate positions properly.

1. Read the first value in the object area by incrementing the pointer to the beginning of the object area, as shown in Figure 35.

```
pObjArea = pOutStructure + FLG_HEADER_SIZE +
           (FLG_DEFINITION_SIZE × number_of_properties)
```

Figure 35. Moving the pointer to the beginning of the object area

   FLG_HEADER_SIZE is the length of the header area and FLG_DEFINITION_SIZE is the length of a record in the definition area.
2. Check the data type and data length for the property this value belongs to in the definition area.

   For CHAR or TIMESTAMP Read in a value that is the length specified in the definition area.

   For VARCHAR or LONG VARCHAR

   a. Read the first 8 characters for this value to determine the length of the value.
   b. Move the pointer 8 bytes to read the value itself.

   Move to the next value in the object area by adding the actual length of the current value to the pointer as shown in Figure 36.

```
pObjValue = pObjArea + actual_value_length
```

Figure 36. Moving the pointer to the next value

Figure 37 shows how to start at the beginning of the object area, read the length of the VARCHAR value, move the pointer to the beginning of the value itself, then read the value before moving the pointer to the next value.

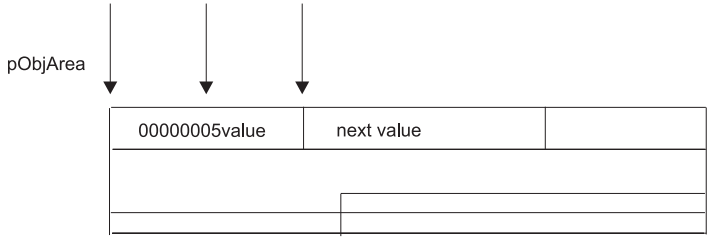

Figure 37. Reading a VARCHAR value in the object area

DG2SAMP.C example of locating a value in an output structure

The DG2SAMP.C program gets an object type name from the user, then issues an FLGListObjTypes call to retrieve a list of object types available in the information catalog database. The program tries to match the external name of an object type specified by the user with a name in the output structure returned by FLGListObjTypes.

Figure 38 shows the format of the output structure produced by an FLGListObjTypes API call.

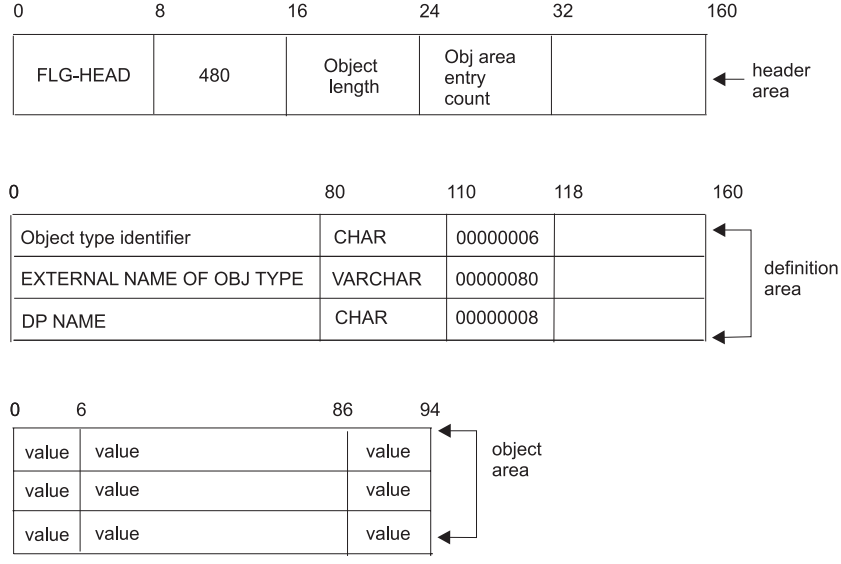

Figure 38. FLGListObjTypes output structure

Getting values from the user and the output structure

Figure 39 shows how the program reads the value specified by the user and calculates its length. It also shows how the program copies values in the output structure into null-terminated strings and calculates the number of sets of values in the object area.

```
gets(pszObjName);      1
ulTypeLen = strlen(pszObjName);      2
memcpy(&pszObjEntryCount, pListStruct->pchHObjEntryCount, FLG_H_OBJAREAENT_LEN);  3
memcpy(&pszDefLength, pListStruct->pchHDefLength, FLG_H_DEFAREA_LEN);             4
ulCount = (atoi(pszObjEntryCount) / (atoi(pszDefLength) / FLG_DEFINITION_SIZE));  5
```

Figure 39. DG2SAMP.C: Getting a value from the user

The code in Figure 39 performs the following steps:

1. Gets the object type name as input from the user
2. Determines the length of the object type name
3. Copies the object entry count into a null-terminated string
4. Copies the definition length into a null-terminated string
5. Calculates the number of sets of values in the object area

Assigning a pointer to the beginning of the object area

The code in Figure 40 assigns a pointer to the beginning of the object area.

In this example, the output for FLGListObjTypes always has the same three properties, so the program does not need to determine the number of properties, the data type, or the data length.

```
ulPosition = 0;                                                    1
pCurrPos = ((UCHAR *)pListStruct + FLG_HEADER_SIZE + ulDefLen);    2
```

Figure 40. DG2SAMP.C: Assigning a pointer to the beginning of the object area

The code in Figure 40 performs the following steps:

1. Sets the position counter to 0.
2. Positions a pointer at the beginning of the object area by adding the length of the header area and definition area to the position of the pointer to the beginning of the output structure (pListStruct).

Moving through the object area

The code in Figure 41 on page 65 moves a pointer through the object area, trying to find an object type name that matches the name given by the user.

```
while (fNotFound && (ulPosition < ulCount))     1
{
  ulPosition = (ulPosition + 1);
  memcpy(&pszObjTypeId, (void *) pCurrPos, FLG_H_OBJTYPID_LEN);        2
  pCurrPos = pCurrPos + FLG_H_OBJTYPID_LEN;                            3
  memcpy(&pszLength, (void *)pCurrPos, FLG_VARIABLE_DATA_LENGTH_LEN); 4
  ulLength = atoi(pszLength);                                          5
  pCurrPos = pCurrPos + FLG_VARIABLE_DATA_LENGTH_LEN;                  6
  strncpy (pszObjectName, (void *)pCurrPos, ulLength);                 7
  pszObjectName[ulLength] = NULLCHAR;                          8

  if (!(strcmp(pszObjName, pszObjectName)))    9
    {
      fNotFound = FALSE;
      printf ("The object type ID for %s is %s.\n\n", pszObjName, pszObjTypeId);
    }
  else
    { // Move temporary pointer to the next object
      pCurrPos = pCurrPos + ulLength + FLG_DPNAME_LEN;   10
    }
}
```

Figure 41. DG2SAMP.C: Matching an object type name with one in the object area

The code in Figure 41 performs the following steps:

1 Checks that the program has not yet found a matching object name, and that the pointer has not yet reached the last set of values in the object area

2 Copies the object type ID of the first object type into `pszObjTypeId`

3 Moves the pointer to the next value, which is the value of the object type name

4 Copies the first 8 characters of the object type name value, which contain the length for this VARCHAR value

5 Converts the length to integer data

6 Moves the pointer past the variable data length to the beginning of the object type name

7 Copies the object type name at the pointer to `pszObjectName`

8 Adds a null character to the end of the object type name to make the value a null-terminated string

9 Compares `pszObjectName` to the object type name specified by the user

10 If the value of `pszObjectName` doesn't match the object type name specified by the user, moves the cursor to the beginning of the next set of values

Reading an output structure resulting from an API call

Chapter 5. The Information Catalog Manager API call syntax

The Information Catalog Manager provides API calls to allow you to use the Information Catalog Manager functions in your own applications.

The API calls are described in alphabetic order. These descriptions include input parameters and structures and output parameters and structures for each API call.

Each API call's description include this information, as appropriate:
- Input parameters
- Input structures
- Output parameters
- Output structures

API call syntax conventions

You must follow certain syntax conventions when using the Information Catalog Manager API calls.

Reading syntax diagrams

The syntax diagrams in this section are written in the form of C language function prototypes.

These function prototypes are defined in the DG2API.H header file, so that you can include (using the `#include` statement) this file in your program without having to specify this function prototype in your own code. "Appendix B. The Information Catalog Manager API header file—DG2APIH" on page 245 lists the data types, function prototypes, and constants defined in the DG2API.H file.

Reason codes are returned as the APIRET data type. APIRET is defined as the unsigned long integer data type in the DG2API.H header file.

Reason codes and extended codes are listed in "Appendix D. Information Catalog Manager reason codes" on page 263.

Using constants defined in DG2API.H in your program

The DG2API.H header file contains structures, typedefs, and commonly used values for the Information Catalog Manager API calls. The function prototypes for the Information Catalog Manager API calls are also included in this file. You can use these constants to help you write your C language program. See

"Appendix B. The Information Catalog Manager API header file—DG2APIH" on page 245 for a list of the constants defined in the Information Catalog Manager API header file.

FLGAppendType

Appends optional properties to an existing object type.

You can append to any object type except the Comments object type, because the Comments object type cannot be extended.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY    FLGAppendType( PFLGHEADERAREA   pObjTypeStruct,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

pObjTypeStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the specifications for one or more properties to be appended for this object type.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGAppendType, you must define the input structure shown in Figure 42 on page 70. This structure contains only the header area and the definition area.

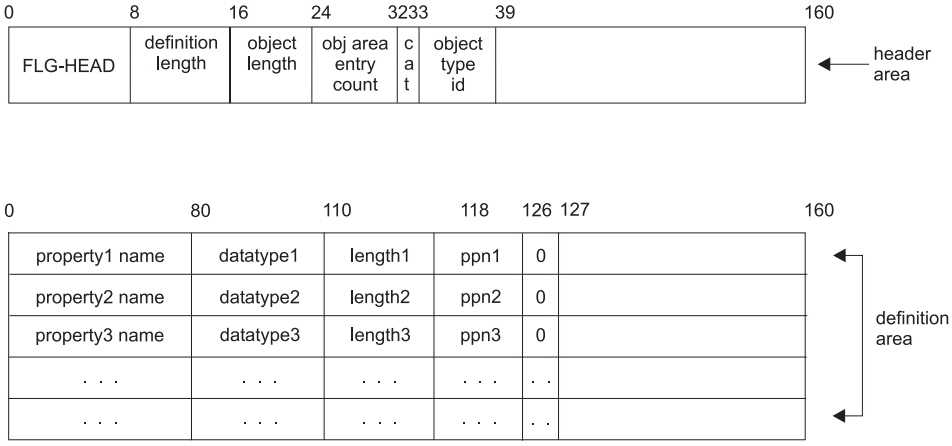

Figure 42. FLGAppendType input structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API input structure" on page 32.

Usage

Restrictions:

You can append to any object type except the Comments object type, because the Comments object type cannot be extended.

If you append a new property that already exists within the object type, the "new" property is treated as a duplicate and FLGAppendType completes successfully with a warning (FLG_WRN_PROPDUP). A property is a duplicate if all of the following match an existing property:

Data type

Data length

Property short name

Value flag

UUI number

Input requirements:

Header area

- The object type ID in bytes 33-38 must exist in the catalog.
- The category specified in byte 32 must match the category of the existing object type specified in bytes 33-38.

Definition area

- The input structure should not contain any previously defined properties for this object type, only new properties that are to be appended.

- Any properties being appended must be optional. Specify the letter O in the value flag field in byte 126.
- Any properties being appended *cannot* be defined as part of the universal unique identifier (UUI); define the field in byte 127 as blank.
- New property names must be unique within the object type.
- New property short names must be unique within the object type. Property short names must follow these rules:
  - Characters must be single-byte character set (SBCS) only.
  - The first character must be an English alphabetic character (A through Z or a through z), @, #, or $.
  - Characters other than the first can be an English alphabetic character (A through Z or a through z), 0 through 9, @, #, $, or _ (underscore).
  - No leading or embedded blanks are allowed.
  - The name cannot be any of the SQL reserved words for the current database. See the documentation for the underlying database for a list of reserved words.
- The total length of all of the properties for an object type must not exceed the environment limit. The limit depends on the maximum limit for a row (including overhead) for your database system. For more information, see the DB2 UDB SQL reference for your database system.

In general:

- The maximum number of properties for an object type is 255 (FLG_MAX_PROPERTIES).
- The maximum number of properties for an object type that can have a data type of LONG VARCHAR is 14 (FLG_MAX_NUM_LONG_VARCHARS).

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGAppendType completes successfully. If FLGAppendType does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 43 on page 72 shows the C language code required to issue the FLGAppendType call. This code appends an additional property named Density to the object type with an object type identifier of 000044.

```
APIRET           rc;              // Declare reason code
PFLGHEADERAREA   pObjTypeStruct;  // Pointer to the input structure
FLGEXTCODE       ExtCode = 0;          // Declare extended code

  .
  .  /* Appending pObjTypeStruct object Type */
  .          /*  by providing object properties     */
  .

rc = FLGAppendType (pObjTypeStruct,
                    &ExtCode);        // Pass pointer to extended code
```

Figure 43. Sample C language call to FLGAppendType

Figure 44 shows the input structure for the FLGAppendType call. The pObjTypeStruct parameter points to this input structure.

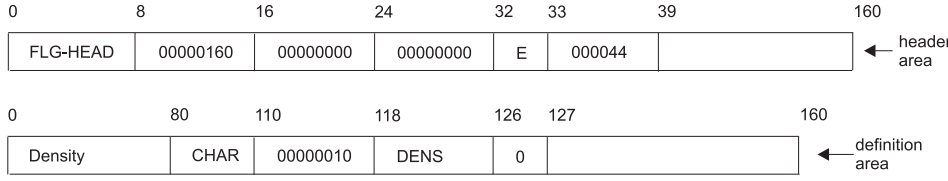

Figure 44. Sample input structure for FLGAppendType

Special error handling

If FLGAppendType encounters a database error, the Information Catalog Manager rolls back the database to the last commit that occurred in your program.

If this rollback is successful, FLGAppendType returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database.

Attention: If this rollback fails, FLGAppendType returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database. In this case, your database could have severe integrity problems, and your program should call FLGTerm to exit the Information Catalog Manager.

Depending on the state of your database, you might need to recover your database using your backed-up database files. For more information about recovering your information catalog database, see the *Information Catalog Manager Administration Guide*.

To prevent the Information Catalog Manager from removing uncommitted changes that are not related to the FLGAppendType error, include FLGCommit calls in your program just before this call.

FLGCommit

Commits all changes made to the information catalog since the unit of work was started or since the last commit point.

Authorization

Administrator or user

Syntax

```
APIRET APIENTRY FLGCommit (PFLGEXTCODE   pExtCode)
```

Parameters

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

APIRET
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Your program should call FLGCommit after making changes to the information catalog database to make these changes permanent. In general, you can have your program call FLGCommit after it makes any change to the database.

The following situations are particularly good opportunities for committing changes to the database:

- After updating a series of related metadata values. To keep related information consistent in the information catalog, your program can issue FLGCommit after making a number of related changes.
- After a set of FLGCreateReg and FLGCreateType calls that completely define a new object type, or a set of FLGDeleteType and FLGDeleteReg calls that completely remove an object type. At this point, you know that your program is not committing a partial object type definition.
- After FLGDeleteTree or FLGDeleteTypeExt calls, because these calls make major changes to your information catalog.
- Before FLGAppendType, FLGCreateReg, FLGCreateType, FLGDeleteType, and FLGDeleteTypeExt calls. These API calls automatically roll back the database when they encounter severe database errors. You can issue FLGCommit calls before one or more of these API calls to prevent the

Information Catalog Manager from removing uncommitted changes that are not related to the database error if a rollback occurs.

- Before an FLGImport call. The Information Catalog Manager rolls back the database when FLGImport encounters errors. Your program should issue FLGCommit before issuing FLGImport to ensure that the Information Catalog Manager does not also roll back uncommitted changes that occurred before the FLGImport call.

Examples

Figure 45 shows the C language code that calls FLGCommit.

```
APIRET        rc;           // Declare reason code from FLGCommit
FLGEXTCODE    ExtCode = 0;       // Declare extended code
  .
  .
rc = FLGCommit(&ExtCode);        // pass the address of
                              // extended code
```

Figure 45. Sample C language call to FLGCommit

Special error handling

If FLGCommit encounters a database error, the Information Catalog Manager rolls back the database to the previous commit that occurred in your program.

If this rollback is successful, FLGCommit returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database.

Attention: If this rollback fails, FLGCommit returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database. In this case, your database could have severe integrity problems, and your program should call FLGTerm to exit the Information Catalog Manager.

Depending on the state of your database, you might need to recover your database using your backed-up database files. For more information about recovering your information catalog database, see the *Information Catalog Manager Administration Guide*.

FLGConvertID

Retrieves the object type ID of an object type given the DP NAME, or the Name of an object instance given the FLGID.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY  FLGConvertID( PSZ             pszInBuffer,
                                PSZ             pszOutBuffer,
                                FLGOPTIONS      Options,
                                PFLGEXTCODE     pExtCode );
```

Parameters

pszInBuffer (PSZ) — input
Points to an input buffer containing either a 16-character system-generated, unique identifier of an object instance (FLGID), or an 8-character short name for an object type (DP NAME).

pszOutBuffer (PSZ) — output
Points to an output buffer containing either an 80-character external name of an object instance, or a 6-character object type ID.

Options (FLGOPTIONS) — input
Choose one of the following options:
FLG_DPNAME
Indicates that the input buffer contains a DP NAME.
FLG_FLGID
Indicates that the input buffer contains an FLGID.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Examples

Figure 46 on page 77 shows the C language code required to issue the FLGConvertID call. This sample code retrieves the object type identifier for a specified object type.

```
APIRET          rc;                 // reason code
PSZ             pszInBuffer;        // pointer to input buffer
PSZ             pszOutBuffer;       // pointer to output buffer
FLGOPTIONS      options=FLG_DPNAME; // option flag
FLGEXTCODE      xc = 0;             // extended code

  .
  .
  .
strcpy (pszInBuffer,"CHARTS");      // object type's DP NAME
  .
  .
  .

rc = FLGConvertID (pszInBuffer,
                   pszOutBuffer,
                   options,
                   &xc);
```

Figure 46. Sample C language call to FLGConvertID

FLGCreateInst

Creates a new instance of the specified object type.

Authorization

Administrator or authorized user (all object types); user (Comments object type only)

Syntax

```
APIRET  APIENTRY    FLGCreateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PSZ              pszFLGID,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

pObjInstStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the property specifications and values for the new object instance.

pszFLGID (PSZ) — output
Points to the 16-character, system-generated ID for the new object instance.

Characters 1-6 of this ID identify the object type of this instance; these characters have the same value as bytes 33 through 38 in the input structure header record.

Characters 7-16 of this ID are the system-generated unique instance identifier.

This returned pszFLGID is used by other API calls when referring to this instance.

pszFLGID is set to NULL if the FLGCreateInst API call is not successful.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGCreateInst, you must define the input structure shown in Figure 47. This structure contains the header area, the definition area, and the object area.

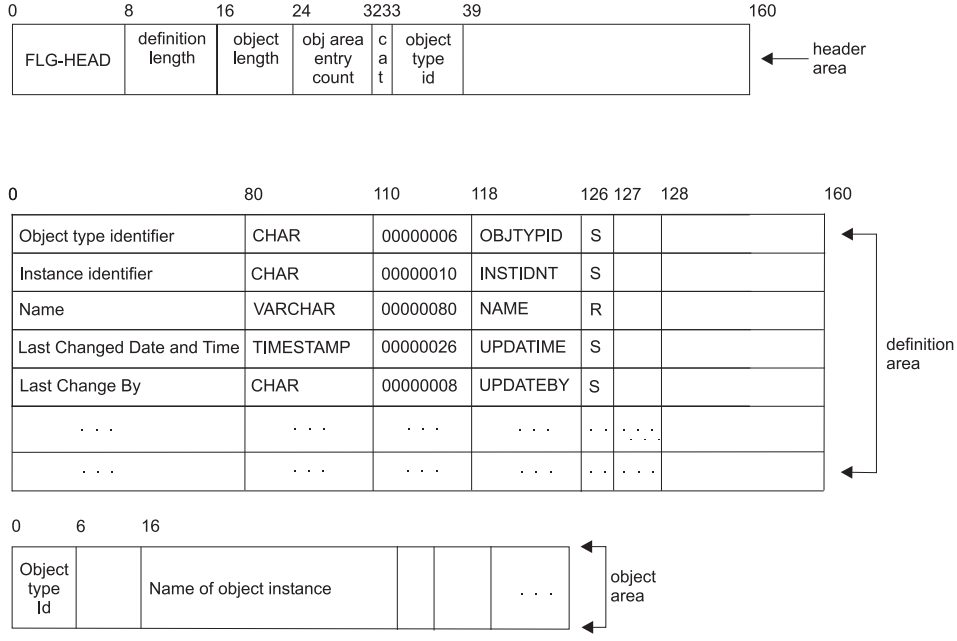

Figure 47. FLGCreateInst input structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API input structure" on page 32.

Usage

Prerequisites

- Before you can create an object instance, the object type must already exist in the information catalog. If it does not, you must register and create the object type by issuing an FLGCreateReg call followed by an FLGCreateType call.
- To issue an FLGCreateInst call, you must have the information about the properties required to define a new instance, either from issuing the FLGCreateType call or from issuing an FLGGetType call to retrieve this information.

Restriction

If you are not authorized to perform object management tasks and you are creating a Comments object instance, you must not change the Creator property value to be other than your logged-on user ID.

Input requirements

Header area:

All of the fields in the header record are required.

Definition area:

The definition area can contain any or all of the defined properties of the object type for which you are creating an object instance. The following rules apply:

- You must first specify all five of the Information Catalog Manager required properties in the following order: OBJTYPID, INSTIDNT, NAME, UPDATIME, and UPDATEBY.
- You must specify all other required (indicated by an R in byte 126) properties.
- The Information Catalog Manager compares all specified properties to the object type definition for the following specifications:

  Data type

  Data length

  Property short name

  Value flag

  UUI number

Object area:

- Values for the following properties must be specified:

  OBJTYPID
  Must be same as Header area, bytes 33 through 38.

  NAME
  Must not be all blank.

  The value of the property NAME does not have to be unique within an object type; you can successfully create duplicate entries. However, when you create duplicate entries, specify some descriptive information as the value of another property to differentiate one object instance from another.

- Values for the following properties are system-generated and must be left blank:
  - INSTIDNT
  - UPDATIME
  - UPDATEBY
- If a value is not specified for a required property (defined with an R in column 126 of the definition area) the appropriate space in the object area must be initialized as follows:

Data type	Initialized to
CHAR	Not-applicable symbol followed by blanks for the length of the property
TIMESTAMP	Set to the largest allowable value: `9999-12-31-24.00.00.000000`
VARCHAR LONG VARCHAR	00000001; the length field, specified in 8 bytes, followed by the not-applicable symbol

- If a value is not specified for an optional property, the appropriate space in the object area must be initialized as follows:

Data type	Initialized to
CHAR TIMESTAMP	Blanks for the entire length of the property
VARCHAR LONG VARCHAR	00000000; the length field, specified in 8 bytes, must be present and set to zero

- The Information Catalog Manager removes all trailing blanks of values in the object area with data types of VARCHAR or LONG VARCHAR, and the length of that area is automatically adjusted.
- The object type in the HANDLES property (when specified) must exist in the information catalog and be a non-Program object type. Any properties specified in the PARMLIST property must be a property of the object type specified in HANDLES. For more information, see "Setting up Programs objects to start programs" on page 25.
- Each object instance must have unique values for the UUI properties. If an object instance already exists with the same UUI values as the object instance being created, an error will occur.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGCreateInst completes successfully. If FLGCreateInst does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 48 on page 82 shows the C language code required to call FLGCreateInst.

This sample code creates a new instance of a Grouping object type.

```
APIRET          rc;                     // Declare reason code
PFLGfEADERAREA  pObjInstStruct;         // Pointer to the input structure
UCHAR           pszFLGID[FLG_ID_LEN+1]; // Returns system-generated ID
FLGEXTCODE      ExtCode = 0;                 // Declare extended code
  .
  /* creating pObjInstStruct object Instance by providing property values */
  .
  .

rc = FLGCreateInst (pObjInstStruct,    // input structure
                    pszFLGID,          // Returned ID of created instance
                    &ExtCode);              // Pass pointer to extended code
```

Figure 48. Sample C language call to FLGCreateInst

Figure 49 shows the input structure for the FLGCreateInst call. The pObjInstStruct parameter points to this structure, which carries the property and value information for the new object instance.

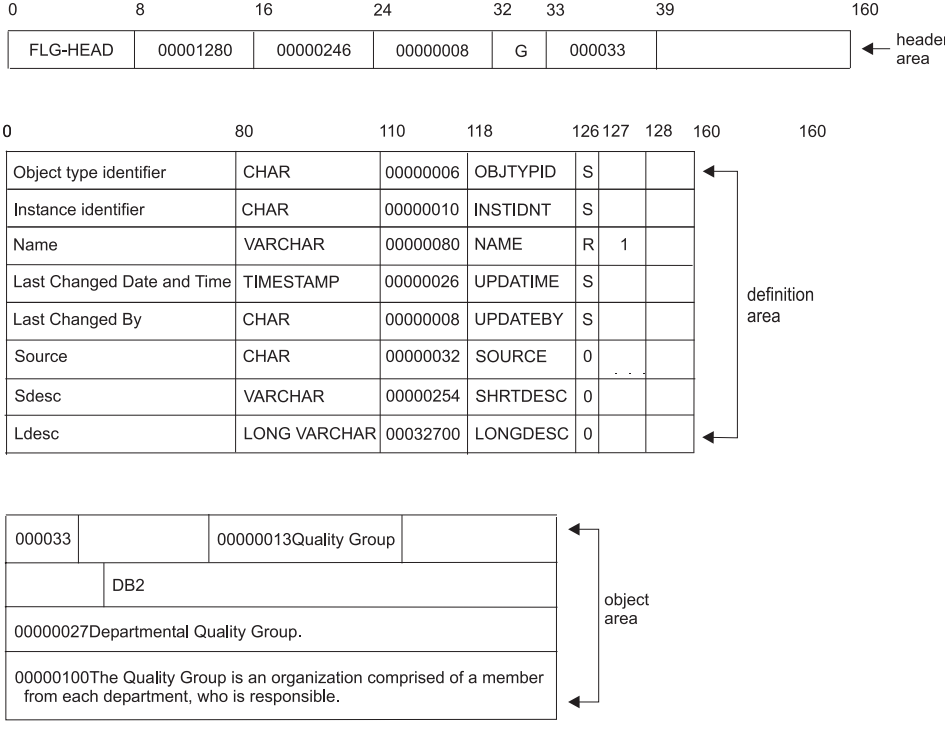

Figure 49. Sample input structure for FLGCreateInst

Notes:

1. Bytes 33 through 38 of the header record contain the object type identifier (000033) that was generated by the FLGCreateReg when this object type was registered. The same value must be specified for the OBJTYPID in the object area. In this example, it appears as the first value in the object area.
2. This object type contains the first five required properties (OBJTYPID, INSTIDNT, NAME, UPDATIME, UPDATEBY) plus three more properties that were added by the user.

FLGCreateReg

Creates registration information in the information catalog for an object type.

This API call does not create the object type itself; it registers the object type so that the object type can be created. The registration information that FLGCreateReg stores in the information catalog includes registration values that describe the object type.

You can register a type for any category except the Program and Attachment categories, because these categories can contain only the Programs and Comments types respectively, which the Information Catalog Manager automatically creates in the information catalog.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY  FLGCreateReg( PFLGHEADERAREA   pObjRegStruct,
                                PSZ             pszIconFileID,
                                PSZ             pszObjTypeID,
                                PFLGEXTCODE     pExtCode );
```

Parameters

pObjRegStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the property specifications and values of the new object type registration.

pszIconFileID (PSZ) — input
Contains the drive, directory path, and file name of the file that contains the icon for the new object type registration. If this parameter is NULL, then no icon is associated with the new object type registration.

pszObjTypeID (PSZ) — output
Points to the 6-character, system-generated unique identifier (object type ID) of the registered object type.

This returned ObjTypeID is used by other API calls to identify the object type. It is set to NULL if the object type is not registered successfully.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGCreateReg, you must define the input structure shown in Figure 50. This structure contains the header area, the definition area, and the object area.

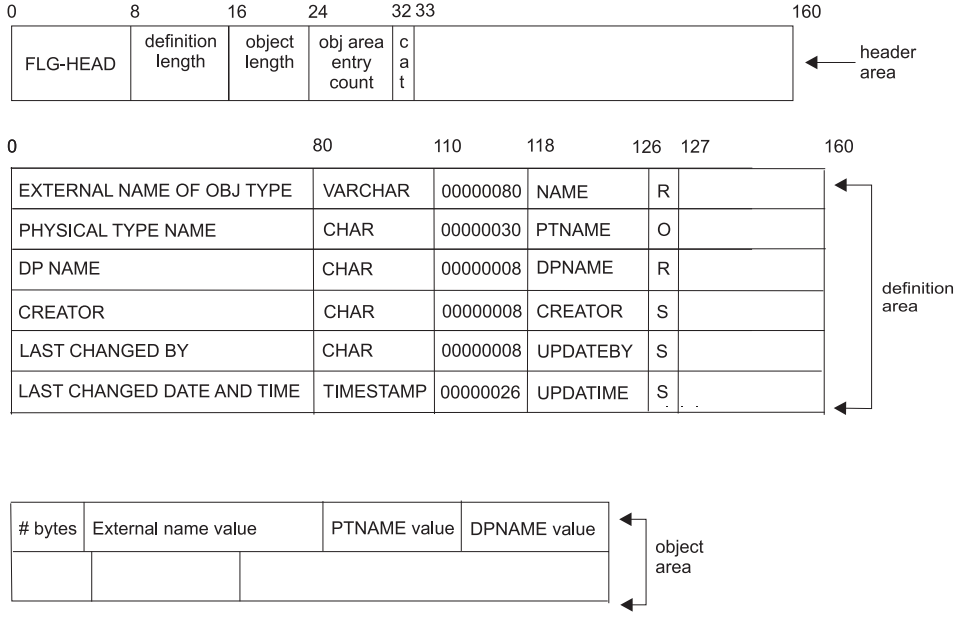

Figure 50. FLGCreateReg input structure

In the input structure, you must specify these six properties in the definition area in the order shown.

If you are using a version of the Information Catalog Manager other than English, the names of these required properties are translated, and are returned in the output structure of FLGInit.

Table 15

Table 15. Properties required for object type registrations

Property short name	Property name[1]	Description	Specify value in object area?
NAME	EXTERNAL NAME OF OBJ TYPE	80-byte name of the object type; can be later modified.	Required.

Table 15. Properties required for object type registrations (continued)

Property short name	Property name[1]	Description	Specify value in object area?
PTNAME	PHYSICAL TYPE NAME	30-character name of the actual table in the information catalog that contains the object type.	Optional; default value is the value for DPNAME.
DPNAME	DP NAME	8-character short name for the object type.	You must set this value using FLGCreateReg. Required.
CREATOR	CREATOR	8-character user ID of the administrator who creates the object type.	No; the Information Catalog Manager sets this value when FLGCreateType is issued for the object type.
UPDATEBY	LAST CHANGED BY	8-character user ID of the administrator who last modified the object type.	No; the Information Catalog Manager sets this value when FLGAppendType is issued for the object type.
UPDATIME	LAST CHANGED DATE AND TIME	26-character time stamp of the last date and time the object type was modified.	No; the Information Catalog Manager sets this value when FLGCreateType or FLGAppendType is issued for the object type.

Note:

1. The property names in this column apply to English versions of the Information Catalog Manager; if you are using a translated version of the Information Catalog Manager, the property name will also be translated.

For a general explanation of the meanings of the byte offsets, see "The Information Catalog Manager API input structure" on page 32.

Usage

Restrictions

- You cannot register a new object type for the Program category (P), because you cannot add any new Program object types. When you create your information catalog, it includes the only permitted object type ("Programs") of category Program.
- You cannot register a new object type for the Attachment category (A), because you cannot add any new Attachment object types. When you create your information catalog, it includes the only permitted object type ("Comments") of category Attachment.
- To assign an icon to the object type, use FLGManageIcons (see "FLGManageIcons" on page 182).

– After you define the object type using FLGCreateReg, you can issue FLGUpdateReg or FLGManageIcons calls to change the icon that is associated with the object type, or add an icon association if one was not defined originally. You can also use FLGManageIcons to remove an icon from an object type.

Input requirements

Header area:

All of the information shown in the header record in Figure 50 on page 85 is required.

Definition area:

- The definition area must contain definitions for each of the six registration properties shown in Figure 50 on page 85. The definitions for each of these registration properties, except translated property names (see Table 15 on page 85), must be specified exactly as shown.
- Each required property name (bytes 0-80 for each property) could be translated from the English property name shown in Figure 50 on page 85 into any of the supported national languages. The translation of the names of these required properties is returned in the output structure of FLGInit.

Object area:

- For properties defined with an S value in byte 126, leave the values in the object area blank; the Information Catalog Manager ignores any specified values for these properties because the system generates these values when you create or append the object type. These properties are CREATOR, UPDATEBY, and UPDATIME.
- Rules for the PTNAME:
  - The PTNAME of the object type must be unique within the Information Catalog Manager catalog.
  - The Information Catalog Manager maximum length for the value of PTNAME is FLG_PTNAME_LEN (30); however, database constraints can shorten the maximum length in your information catalog environment. See the *Information Catalog Manager Administration Guide* for more information about setting this maximum.
  - If the number of significant characters of the PTNAME, not including trailing blanks, exceeds the maximum allowed for your environment (the value of STOR ENVSIZE returned by FLGInit), the registration request is rejected.
  - Specifying the PTNAME is optional. If you do not specify the PTNAME, then the Information Catalog Manager sets it to the value of DPNAME by default.
  - The restrictions for PTNAME are:

  - Must be SBCS only
  - The first character must be an English alphabetic character (A through Z or a through z), @, #, $
  - Characters other than the first can be an English alphabetic character (A through Z or a through z), 0 through 9, @, #, $, or _ (underscore).
  - No leading or embedded blanks are allowed
  - The PTNAME cannot be any of the SQL reserved words for your database

- The DPNAME of the object type:
  - Must be unique among all the information catalogs in the organization
  - Must be SBCS only
  - The first character must be an English alphabetic character (A through Z or a through z), @, #, or $.
  - Characters other than the first can be an English alphabetic character (A through Z or a through z), 0 through 9, @, #, $, or _ (underscore).
  - No leading or embedded blanks are allowed
  - The NAME value must be unique within the local information catalog.

Output information

The system-generated object type identification is returned in the output parameter pszObjTypeID. When the Information Catalog Manager returns this number, you use this number in subsequent calls, such as FLGDeleteReg or FLGGetReg, to uniquely identify the object type registration.

Controlling updates to your information catalog

If FLGCreateReg does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207). Do not call FLGCommit after FLGCreateReg completes successfully—wait until you complete a call to FLGCreateType.

Examples

Figure 51 on page 89 shows the C language code required to issue the FLGCreateReg call. This sample code creates registration information for a new object type called MYIMAGE that belongs to the Elemental category.

```
APIRET         rc;                  // Declare reason code
PFLGHEADERAREA pObjRegStruct;       // Pointer to the input structure
UCHAR          pszIconFileID[FLG_ICON_FILE_ID_MAXLEN+1]; // Path/File name of ICON
UCHAR          pszObjTypeID[FLG_OBJTYPID_LEN+1];      // Returned system-generated ID
FLGEXTCODE     ExtCode = 0;       // Declare extended code
  .
  .  /* creating pObjRegStruct object Type Registration by providing values  */
  .
strcpy (pszIconFileID,"Y:\\FLGICON2.ICO");

rc = FLGCreateReg (pObjRegStruct,   // input structure
                   pszIconFileID,     // Path/File name of file containing the ICON
                   pszObjTypeID,      //  Returned id of registered object type
                   &ExtCode);              // Pass extended code pointer
```

Figure 51. Sample C language call to FLGCreateReg

Figure 52 shows the input structure for the FLGCreateReg call. The pObjRegStruct pointer points to this structure, which carries the property and value information needed for registration of the new object type.

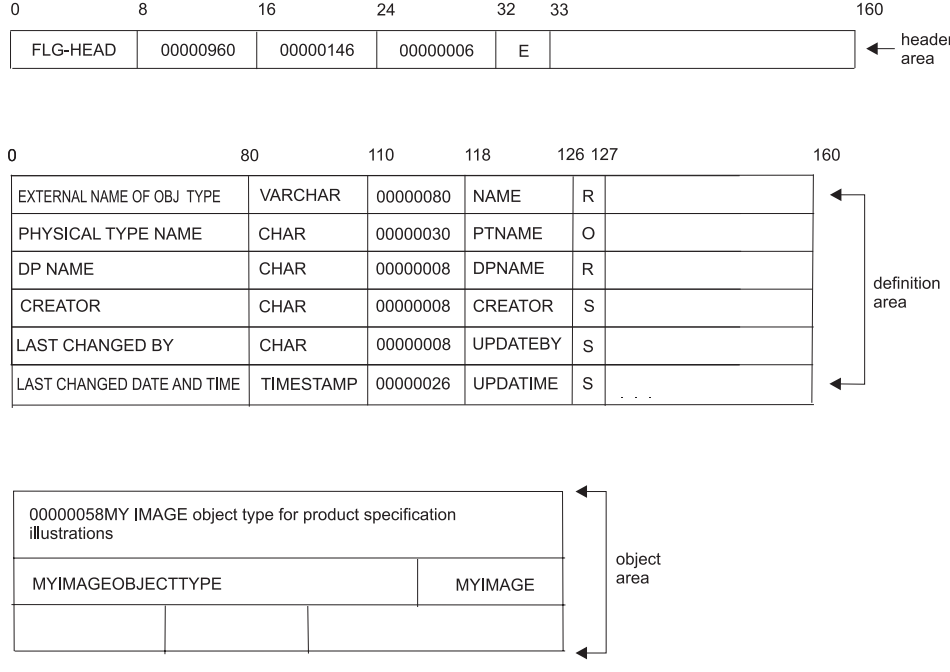

Figure 52. Sample input structure for FLGCreateReg

Special error handling

If FLGCreateReg encounters a database error, the Information Catalog Manager rolls back the database to the last commit that occurred in your program.

If this rollback is successful, FLGCreateReg returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database.

Attention: If this rollback fails, FLGCreateReg returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database. In this case, your database could have severe integrity problems, and your program should call FLGTerm to exit the Information Catalog Manager.

Depending on the state of your database, you might need to recover your database using your backed-up database files. For more information about recovering your information catalog database, see the *Information Catalog Manager Administration Guide*.

To prevent the Information Catalog Manager from removing uncommitted changes that are not related to the FLGCreateReg error, include FLGCommit calls in your program just before this call.

FLGCreateType

Creates a new user-defined object type.

The Administrator can create a type for any category except the Program and Attachment categories, because these categories can contain only the Programs and Comments types respectively, which the Information Catalog Manager automatically creates in the information catalog.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGCreateType( PFLGHEADERAREA   pObjTypeStruct,
                                  PFLGEXTCODE      pExtCode );
```

Parameters

pObjTypeStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the specifications of the properties for this object type.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGCreateType, you must define the input structure shown in Figure 53 on page 92. This structure contains only the header area and the definition area.

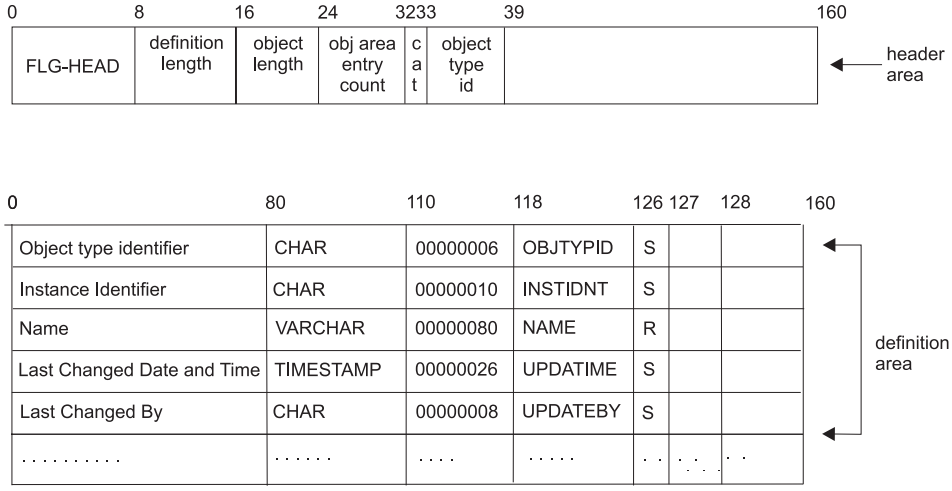

Figure 53. FLGCreateType input structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API input structure" on page 32.

Usage

Prerequisites:

Before you can call the FLGCreateType API to create a new object type, you need to call the FLGCreateReg API to register this new type.

You need to specify the object type identifier returned by the FLGCreateReg API when you call FLGCreateType.

Input requirements:

Header area

The object type that you specify in the header of the input structure must be registered, but not yet created.

Definition area

- The first five properties defined in the definition area must be for these five Information Catalog Manager required properties: OBJTYPID, INSTIDNT, NAME, UPDATIME, and UPDATEBY. If these properties are not in this order, the create will fail.

  OBJTYPID
  Unique system-generated identifier (ID) for the object type

  INSTIDNT
  Unique system-generated ID for an object instance

  NAME
  User-specified name for an object

UPDATIME
System-generated time stamp of when the object was last updated

UPDATEBY
System-generated user ID of the administrator or user who last updated the object

- Rules for the required properties:
  - The data type, length, property short name, and value flag (vf) of each of these required properties are fixed and must be specified exactly as shown in Figure 53 on page 92.
  - The UUI sequence (us) is fixed as blank for each of the four system-generated (S) properties, but can be 1, 2, 3, 4, 5, or blank for the NAME property.
  - The 80-byte property name is fixed, but it is translated for the supported national language versions. The translation of the names of these required properties is returned in the output structure of FLGInit.
- The total number of properties in the definition must not exceed FLG_MAX_PROPERTIES (255).
- The total number of properties in the definition that have a data type of LONG VARCHAR must not exceed the Information Catalog Manager limit of FLG_MAX_NUM_LONG_VARCHARS (14).
- Rules for the UUI:
  - At least one UUI property must be defined for each object type created.
  - Within the object type, you must start the UUI numbering with 1 and not skip any values. For example, in an object type, a set of UUI sequence values of 1, 2, and 3 is valid, but 2, 3, and 5 is not.
  - You cannot specify the same UUI sequence number more than once in the same object type.
  - Any property specified as a UUI must not exceed 254 bytes in length.
  - Any property specified as a UUI must be a required property (″R″ value-flag, column 126).
  - Any property specified as a UUI (column 127) must not have a data type of LONG VARCHAR.
  - You should define the UUI properties so that each instance of this object type can be uniquely identified. You should be able to use these properties to identify a single instance of this object type, even if instances of this object type exist in several related information catalogs.

- In addition to the required properties, the user can add more properties to tailor the created object type to the needs of the business.

  The order of these additional properties in the definition area does not matter because the Information Catalog Manager uses the property short names in bytes 118-125 as a key to ensure that all required properties are always specified. However, the order of the properties in the definition area for the FLGCreateType call is the order in which the Information Catalog Manager returns the properties to the calling application.
- New property names must be unique within the object type.
- New property short names must be unique within the object type.
- If a property belongs to an object type that is shared among two or more related information catalogs, and you plan to import and export the Information Catalog Manager data to share information, then the values for data type, data length, property short name, value flag, and UUI sequence must be same as for the same object type in the other information catalogs. The property name can be different.
- Property short names must follow these rules:
  - Must be (SBCS) only.
  - The first character must be an English alphabetic character (A through Z or a through z), @, #, or $.
  - Characters other than the first can be an English alphabetic character (A through Z or a through z), 0 through 9, @, #, $, or _ (underscore).
  - No leading or embedded blanks are allowed.
  - Cannot be any of the SQL reserved words for the current database.
- The total length of all of the properties for an object type must not exceed the row limit for the underlying database. See the documentation for the underlying database for information about calculating the row length.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGCreateType completes successfully. If FLGCreateType does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 54 on page 95 shows the C language code required to issue the FLGCreateType API call.

This sample code creates a new object type. This new object type is of the Elemental category, as indicated by E in the structure header area, with an object type ID of 000044. Along with the Information Catalog Manager-required properties, this object type contains three additional required properties: imagecolor, imagesize, and description.

```
APIRET          rc;              // Declare reason code
PFLGHEADERAREA  pObjTypeStruct;  // Pointer to the input structure
FLGEXTCODE      ExtCode=0;           // Declare extended code
  .

  .  /* creating pObjTypeStruct object type by */
     /* providing object type's properties  */

  .

rc = FLGCreateType (pObjTypeStruct,
                    &ExtCode);         // Pass pointer to extended code
```

Figure 54. Sample C language call to FLGCreateType

Figure 55 shows the input structure for the FLGCreateType API call. The pObjTypeStruct pointer points to the structure that carries the property information for the new object type.

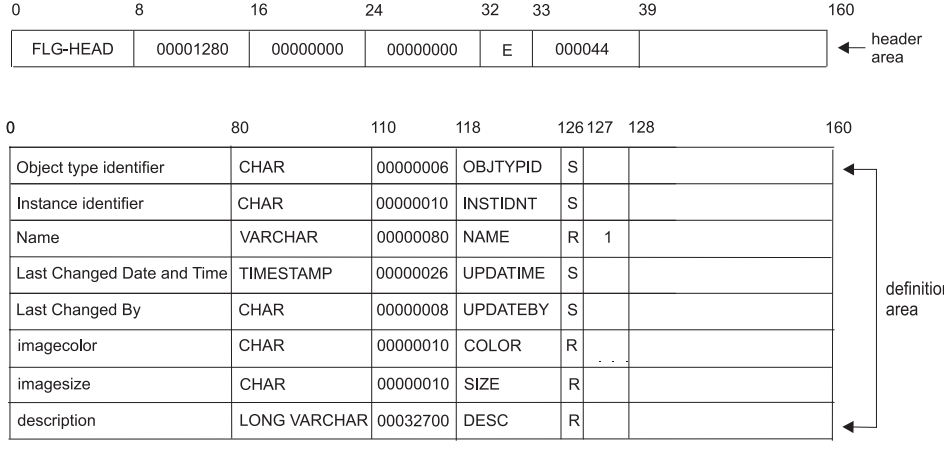

Figure 55. Sample input structure for FLGCreateType

Special error handling

If FLGCreateType encounters a database error, the Information Catalog Manager rolls back the database to the last commit that occurred in your program.

If this rollback is successful, FLGCreateType returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database.

Attention: If this rollback fails, FLGCreateType returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database. In this case, your database could have severe integrity problems, and your program should call FLGTerm to exit the Information Catalog Manager.

Depending on the state of your database, you might need to recover your database using your backed-up database files. For more information about recovering your information catalog database, see the *Information Catalog Manager Administration Guide*.

To prevent the Information Catalog Manager from removing uncommitted changes that are not related to the FLGCreateType error, include FLGCommit calls in your program just before the call to FLGCreateReg for the object type you are creating.

FLGDeleteInst

Deletes a single, specified object instance of an object type.

Authorization

Administrator or authorized user (all object types); user (Comments object type only)

Syntax

```
APIRET  APIENTRY    FLGDeleteInst( PSZ             pszFLGID,
                                   PFLGEXTCODE     pExtCode );
```

Parameters

pszFLGID (PSZ) — input
Points to the 16-character, system-generated unique identifier of the instance to be deleted.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Prerequisites

The value specified for the pszFLGID input parameter must exist.

Restrictions

If you are a user who has not been authorized to perform object management tasks, you can only delete Comments instances for which the value of your logged-on user ID is the same as the value of the Creator property.

Rules for object instances with relationships

For instances participating in Attachment relationships:

- If the instance has one or more associated Comments instances, then all the Comments instances and all such relationships are deleted when the object instance itself is deleted.

- If the instance is a Comments instance in an Attachment relationship, then all such relationships are deleted when the Comments object instance itself is deleted.

For instances that are contained or containers:

- If the instance is a container, you must delete all relationships with contained object instances before deleting the instance using FLGDeleteInst. If you want to delete an instance that is a container and all relationships with contained object instances, you can use FLGDeleteTree instead (see "FLGDeleteTree" on page 102).
- If the instance is contained by another object, you can delete the instance without first deleting the relationship with the container object. Both the relationship and the instance itself are automatically deleted.

For instances participating in Contact relationships:

- If the instance participates in any Contact relationship, then all such relationships are deleted when the object instance itself is deleted.
- If the instance is a Contact in a Contact relationship, then all such relationships are deleted when the Contact object instance itself is deleted.

For instances participating in link relationships:

If the instance participates in link relationships, then all such relationships are deleted when the object instance itself is deleted.

For Programs instances associated with non-Program object types:

A Programs instance can be deleted at any time without affecting any associated object types.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGDeleteInst completes successfully. If FLGDeleteInst does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 56 on page 99 shows the C language code required to issue the FLGDeleteInst call. This sample code deletes an object instance.

```
APIRET        rc;                          // Declare reason code
UCHAR         pszFLGID[FLG_ID_LEN + 1];    // Unique instance identifier
FLGEXTCODE    ExtCode = 0;              // Declare extended code
  .
  .  /* Get FLGID for object instance using FLGSearch. */
  .
strcpy (pszFLGID,"0000330000001234");
rc = FLGDeleteInst (pszFLGID,              // Instance ID
                    &ExtCode);
```

Figure 56. Sample C language call to FLGDeleteInst

FLGDeleteReg

Deletes a specific object type registration from the information catalog.

You can delete registration for a type of any category except the Program and Attachment categories, because the Information Catalog Manager provides these categories when it creates the information catalog.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY  FLGDeleteReg( PSZ           pszObjTypeID,
                                PFLGEXTCODE   pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character, system-generated unique identifier (object type ID) of the object type for which you are deleting the registration.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

This action does not delete the object type itself; it deletes the registration for the object type.

Restrictions

The value for the input parameter pszObjTypeID must exist for an object registration in the information catalog.

Before you can delete the registration for the object type, the object type itself must not exist. If the object type exists, you must delete the object type using FLGDeleteType.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74

page 74) after FLGDeleteReg completes successfully. If FLGDeleteReg does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 57 shows the C language code required to invoke the FLGDeleteReg API call. This sample code deletes the registration information for an object type from the information catalogs.

```
APIRET       rc;           // Declare reason code
UCHAR        pszObjTypeID[FLG_OBJTYPID_LEN + 1];
FLGEXTCODE   ExtCode = 0;       // Declare extended code
  .
  .  /* Get object type ID using FLGConvertID. */
  .
strcpy (pszObjTypeID,"000044");
rc = FLGDeleteReg (pszObjTypeID,    // object type ID
                   &ExtCode);
```

Figure 57. Sample C language call to FLGDeleteReg

The example shown in Figure 57 assumes that the object type ID that was returned when you created the object registration (using FLGCreateReg) was 000044.

FLGDeleteTree

Deletes a specific instance of a Grouping object type, all Comments instances attached to it, and all ATTACHMENT, CONTACT, CONTAIN, and LINK relationships in which it participates. Optionally also deletes all object instances contained in the Grouping category object instance, all Comments instances attached to them, and all ATTACHMENT, CONTACT, and LINK relationships in which they participate.

Authorization

Administrator or authorized user

Syntax

```
APIRET  APIENTRY   FLGDeleteTree( PSZ              pszFLGID,
                                  FLGOPTIONS       Options,
                                  PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

Parameters

pszFLGID (PSZ) — input
Points to the 16-character, system-generated unique identifier of the Grouping category instance (container) to be deleted.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

Options (FLGOPTIONS) — input
Choose one of the following deletion options:

FLG_DELTREE_ALL
Deletes a Grouping category object instance, all Comments instances attached to it, and all ATTACHMENT, CONTACT, and LINK relationships in which it participates. Deletes all object instances contained in the Grouping category object instance, all Comments instances attached to them, and all ATTACHMENT, CONTACT, and LINK relationships in which they participate. See Figure 58 on page 104 through Figure 60 on page 105 for a graphical illustration of this option.

FLG_DELTREE_REL
Deletes a Grouping category object instance, all Comments instances attached to it, and all ATTACHMENT, CONTACT, and LINK relationships in which it participates. Deletes the underlying tree structure of CONTAIN relationships. See Figure 58 on page 104 through Figure 60 on page 105 for a graphical illustration of this option.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure containing a list of deleted object instances.

This output structure contains the 16-character FLGID of each deleted object instance.

If this parameter is NULL, no output structure will be returned. If there is no output structure, then the pointer to the output structure is set to NULL.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Prerequisite

The specified object instance ID (FLGID) must exist.

Restriction

Object instances that are contained by other Grouping objects than the one being deleted (as illustrated in Figure 59 on page 104) are not deleted.

Freeing memory allocated for an output structure

If FLGDeleteTree returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGDeleteTree completes successfully. If FLGDeleteTree does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 58 on page 104 through Figure 60 on page 105 illustrate the effects of the two delete options. Figure 58 on page 104 shows an information catalog with three grouping objects `A`, `Z`, and `Y`. Object `B` will be deleted using FLGDeleteTree.

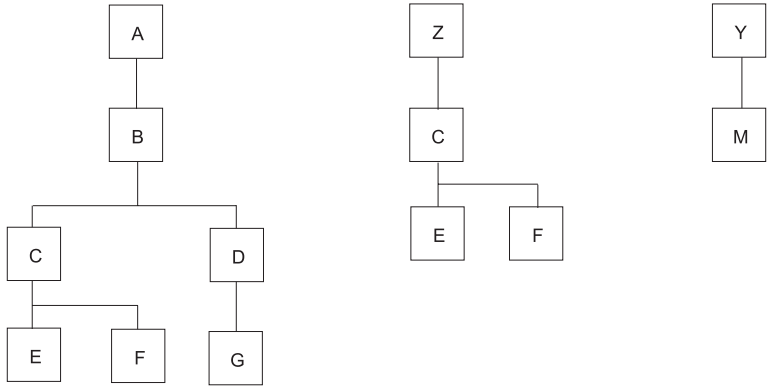

Figure 58. Sample information catalog before deletions

Using the FLG_DELTREE_REL option, object instance B and some relationships under B are deleted. Object C and its containees are not touched because C is contained by another tree, Z. Object D is not contained by any other object and is therefore subject to the cascading effect.

Figure 59 illustrates the information catalog after B is deleted.

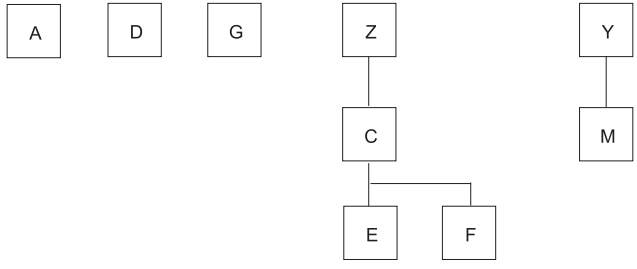

Figure 59. Example of the FLG_DELTREE_REL option

Using option FLG_DELTREE_ALL, object instance B and some instances under it are deleted from the catalog. Object instance C and its containees are kept, because it is also contained by Z.

Figure 60 on page 105 shows the information catalog after B is deleted using the FLG_DELTREE_ALL option.

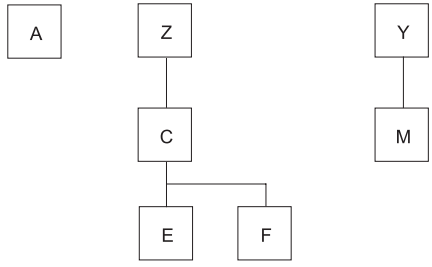

Figure 60. Example of the FLG_DELTREE_ALL option

Figure 61 shows the C language code required to issue the FLGDeleteTree call. This sample code deletes the DEPT001 Grouping category object instance, all Comments instances attached to it, and all ATTACHMENT, CONTACT, and LINK relationships in which it participates. The sample code also deletes all object instances contained in DEPT001 object instance, all Comments instances attached to them, and all ATTACHMENT, CONTACT, and LINK relationships in which they participate.

```
APIRET       rc;                              // Declare reason code
FLGOPTIONS   ulOptMask=0;
UCHAR        pszFLGID[FLG_ID_LEN + 1];
PFLGHEADERAREA  pDelStruct=NULL;
FLGEXTCODE   xc = 0;                          // Declare extended code

  .
  .  set value for pszFLGID
  .

ulOptMask = ulOptMask | FLG_DELTREE_ALL;   // delete whole tree
rc = FLGDeleteTree (pszFLGID, ulOptMask,
                    &pDelStruct, &xc);
```

Figure 61. Sample C language call to FLGDeleteTree

Figure 62 on page 106 shows the output structure for the FLGDeleteTree call.

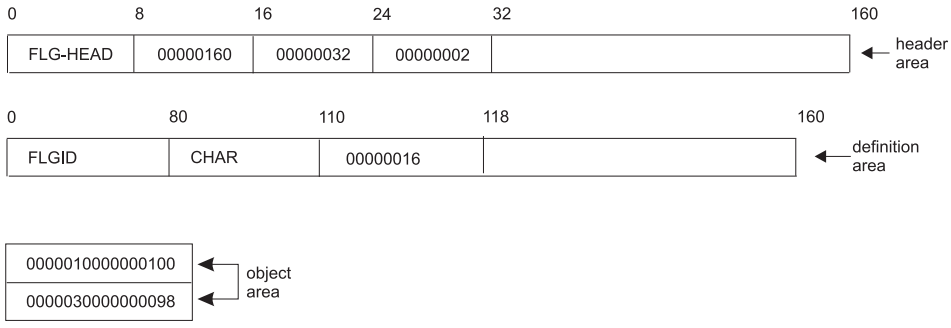

Figure 62. Sample output structure for FLGDeleteTree

FLGDeleteType

Deletes a user-defined object type.

You can delete an object type of any category except the Program and Attachment categories, because the Information Catalog Manager provides these categories when it creates the information catalog.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGDeleteType( PSZ             pszObjTypeID,
                                  PFLGEXTCODE     pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character system-generated unique identifier (object type ID) for the object type to be deleted.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Prerequisites:

The object type ID specified as the input parameter must exist.

No object instances can exist for the object type. If instances of an object type exist, you must delete them using FLGDeleteInst before you can delete the object type. You can either delete the instances individually using FLGDeleteInst or delete several instances at once by importing a tag language file.

You cannot delete the Programs object type that was automatically created in your information catalog. However, an object type can be deleted if it is related to one or more Program instances. The Program instances are automatically updated to clear the values for the HANDLES and PARMLIST properties.

You cannot delete the Comments object type that was automatically created in your information catalog.

Controlling updates to your information catalog

If FLGDeleteType does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207). Do not call FLGCommit after FLGDeleteType completes successfully—wait until you complete a call to FLGDeleteReg.

Examples

Figure 63 shows the C language code required to invoke the FLGDeleteType API call. This sample code deletes an object type from the information catalog.

```
APIRET       rc;           // Declare reason code
UCHAR        pszObjTypeID[FLG_OBJTYPID_LEN + 1];
FLGEXTCODE   ExtCode = 0;       // Declare extended code

  .
  .  /* Get the object type ID of MYIMAGE using FLGConvertID. */
  .

rc = FLGDeleteType (pszObjTypeID,
                    &ExtCode);
  .
  . // if (rc == 0)
  . //   rc = FLGDeleteReg (pszObjTypeID, &ExtCode);
  . // if (rc == 0)
  . //   rc = FLGCommit (&ExtCode);
  . // else
  . //   rc = FLGRollback (&ExtCode);
  .
```

Figure 63. Sample C language call to FLGDeleteType

If instances of MYIMAGE exist, you must delete them before you can delete the MYIMAGE object type. You can either delete the instances individually using the administrator user interface or delete several instances at once by importing a tag language file.

Special error handling

If FLGDeleteType encounters a database error, the Information Catalog Manager rolls back the database to the last commit that occurred in your program.

If this rollback is successful, FLGDeleteType returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database.

Attention: If this rollback fails, FLGDeleteType returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database. In this case, your database could have severe integrity problems, and your program should call FLGTerm to exit the Information Catalog Manager.

Depending on the state of your database, you might need to recover your database using your backed-up database files. For more information about recovering your information catalog database, see the *Information Catalog Manager Administration Guide*.

To prevent the Information Catalog Manager from removing uncommitted changes that are not related to the FLGDeleteType error, include FLGCommit calls in your program just before this call.

FLGDeleteTypeExt

Deletes a user-defined object type and instances of that object type, any Comments objects attached to those instances, and any relationships in which those instances participate. Also deletes the object type registration.

You can delete an object type of any category except the Program and Attachment categories, because the Information Catalog Manager provides these categories when it creates the information catalog.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGDeleteTypeExt( PSZ             pszObjTypeID,
                                     PFLGEXTCODE     pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character system-generated unique identifier (object type ID) for the object type to delete.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Prerequisites:

The object type ID specified as the input parameter must exist.

Restrictions:

FLGDeleteTypeExt does not delete Grouping category object instances that contain instances of objects of a different object type. If such Grouping category instances exist, you must delete them using FLGDeleteTree before you can delete the object type.

You cannot delete the Programs object type that was automatically created in your information catalog. However, an object type can be deleted if it is

related to one or more Program instances. The Program instances are automatically updated to clear the values for the HANDLES and PARMLIST properties.

You cannot delete the Comments object type that was automatically created in your information catalog.

Controlling updates to your information catalog

Because FLGDeleteTypeExt deletes all instances of an object type along with the object type, before calling FLGDeleteTypeExt you might want to search for objects of a particular type to ensure that you do not want to retain any of the existing objects of the object type you want to delete.

To keep your program as synchronized as possible with your information catalog, include a call to FLGCommit (see "FLGCommit" on page 74) after FLGDeleteTypeExt completes successfully. If FLGDeleteTypeExt does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 64 shows the C language code required to issue the FLGDeleteTypeExt call. This sample code deletes from the information catalog the MYIMAGE object type, all instances of the MYIMAGE object type, all comments attached to instances of the MYIMAGE object type, all relationships in which the MYIMAGE instances participate, and the registration for the MYIMAGE object type.

```
APIRET       rc;           // Declare reason code
UCHAR        pszTypeID[FLG_OBJTYPID_LEN+1];
FLGEXTCODE   xc = 0;       // Declare extended code

  .
  .  /* processing */
  .

rc = FLGDeleteTypeExt (pszTypeID,
                       &xc);
```

Figure 64. Sample C language call to FLGDeleteTypeExt

Special error handling

If FLGDeleteTypeExt encounters a database error, the Information Catalog Manager rolls back the database to the last commit that occurred in your program.

If this rollback is successful, FLGDeleteTypeExt returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE. The extended code

contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database.

Attention: If this rollback fails, FLGDeleteTypeExt returns the reason code FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL. The extended code contains the SQL code for the database error that prompted the Information Catalog Manager to roll back the database. In this case, your database could have severe integrity problems, and your program should call FLGTerm to exit the Information Catalog Manager.

Depending on the state of your database, you might need to recover your database using your backed-up database files. For more information about recovering your information catalog database, see the *Information Catalog Manager Administration Guide*.

To prevent the Information Catalog Manager from removing uncommitted changes that are not related to the FLGDeleteTypeExt error, include FLGCommit calls in your program just before this call.

FLGExport

Retrieves metadata from the information catalog and translates it to tag language in a file.

Authorization

Administrator or authorized user

Syntax

```
APIRET  APIENTRY   FLGExport( PSZ              pszTagFileID,
                              PSZ              pszLogFileID,
                              PSZ              pszIcoPath,
                              PFLGHEADERAREA   pListStruct,
                              PFLGEXTCODE      pExtCode );
```

Parameters

pszTagFileID (PSZ) — input
Points to the name of the output tag language file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a file allocation table (FAT) or HPFS file. The target drive for this file can be either a fixed or removable disk. If you type only the file name, the Information Catalog Manager places the tag language file on the drive and path pointed to by the DGWPATH environment variable.

The target tag language file must not exist; the Information Catalog Manager does not overwrite existing tag files.

The file name and extension (excluding the drive and directories) cannot exceed 240 characters. The entire tag language file ID cannot exceed 259 characters.

pszLogFileID (PSZ) — input
Points to the name of the log file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a FAT or HPFS file. The target drive for the log file must be a fixed disk. The log file ID cannot exceed 259 characters. If you specify only a file name, the Information Catalog Manager places the log file on the drive and path pointed to by the DGWPATH environment variable.

If the log file specified in this parameter does not exist, a new file is created. If the log file specified in this parameter already exists, then the FLGExport API call appends to it.

pszIcoPath (PSZ) — input
Points to the specification of the path containing the OS/2® or Windows icon files.

This parameter is optional. If this parameter is NULL, no icon files are exported.

This parameter contains the drive and directories and must be valid for a FAT or HPFS file. This parameter cannot be longer than 246 characters.

If this parameter is specified, the target drive for the icon files must be a fixed disk.

pListStruct (PFLGHEADERAREA) — input
Points to an input structure containing the list of objects to be exported and the export options.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGExport, you must define the input structure shown in Figure 65 on page 115. This structure contains the header area, the definition area, and the object area.

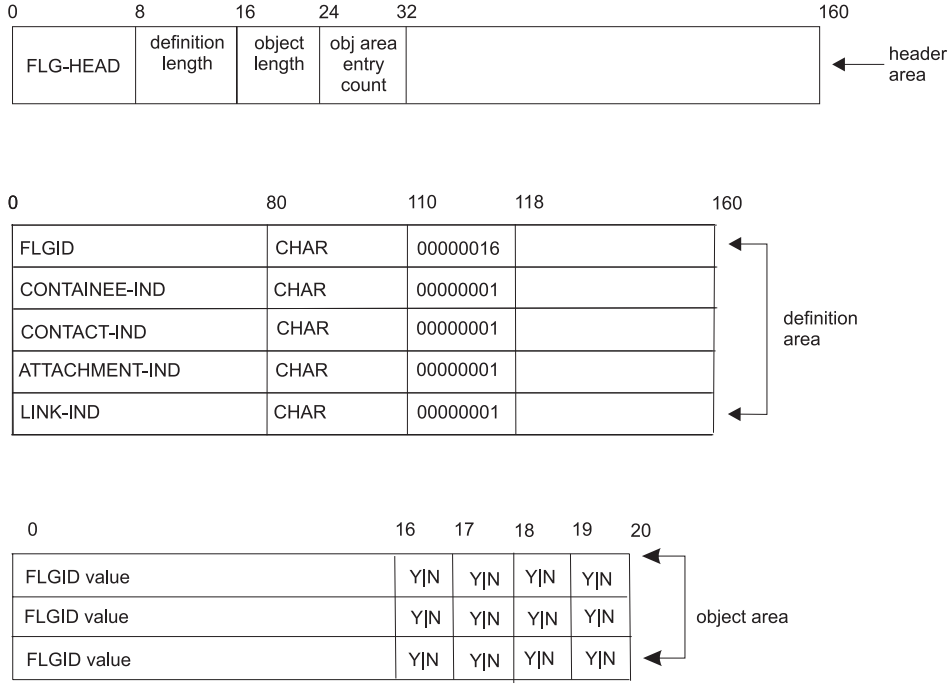

Figure 65. FLGExport input structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API input structure" on page 32.

Usage

Input structure

The definition area for the FLGExport input structure must be specified exactly as shown in Figure 65.

The input structure for FLGExport contains the following information:

FLGID

16-character, system-generated unique identifier of the instance to be exported.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

You can export any Information Catalog Manager object instances.

CONTAINEE-IND

1-character indicator (Y | N) that specifies whether the

Information Catalog Manager exports all objects contained by this object. This indicator applies only to Grouping objects and is ignored for all other objects.

CONTACT-IND
1-character indicator (Y | N) that specifies whether the Information Catalog Manager exports all associated Contact objects of Grouping and Elemental objects. This indicator applies only to Grouping and Elemental objects and is ignored for all other objects.

ATTACHMENT-IND
1-character indicator (Y | N) that specifies whether the Information Catalog Manager exports all Attachment objects attached to the specified object instance. This indicator is ignored if the specified object is an Attachment object.

LINK-IND
1-character indicator (Y | N) that specifies whether the Information Catalog Manager exports all Grouping and Elemental object instances linked with the specified object instance. This indicator applies only to Grouping and Elemental objects and is ignored for all other objects.

Generated tag language file

FLGExport generates a tag language file that contains tags for each object instance exported. Depending on what you specify for the indicators, object instances are exported as shown in Table 16.

Table 16. Object instances exported to tag language file for indicator combinations

Indicator value				
CONTAINEE	**CONTACT**	**ATTACHMENT**	**LINK**	**Exports:**
Y	Y	Y	Y	a through j
Y	Y	Y	N	a, b, c, d, g, h, i, j
Y	Y	N	Y	a, b, e, f, g, h
Y	Y	N	N	a, b, g, h
Y	N	Y	Y	a, b, c, d, e, f
Y	N	Y	N	a, b, c, d
Y	N	N	N	a, b
Y	N	N	Y	a, b, e, f
N	Y	Y	Y	a, c, e, g, i
N	Y	Y	N	a, c, g, i
N	Y	N	Y	a, e, g

Table 16. Object instances exported to tag language file for indicator combinations (continued)

Indicator value				
CONTAINEE	**CONTACT**	**ATTACHMENT**	**LINK**	**Exports:**
N	N	Y	Y	a, c, e
N	N	Y	N	a, c
N	N	N	Y	a, e
N	N	N	N	a only

Notes:

a Specified object instance
b Object instances contained by a
c Comments attached to a
d Comments attached to b
e Links for a
f Links for b
g Contacts for a
h Contacts for b
i Comments attached to g
j Comments attached to h

FLGExport generates frequent COMMIT tags in the tag language file.

FLGExport places a copy of the icon associated with each object type in the specified icon path. FLGExport does not export the default category icons if no other icon is associated with the object type. The name of the exported icon file is the object type DP NAME (short name) with an extension of .ICO for OS/2 icons or .ICW for Windows icons.

Linking your VisualAge C++ program when it exports metadata to diskettes

If your C language program issues an FLGExport call that exports the Information Catalog Manager information to diskettes, link your program with an application type of WINDOWAPI so that the Information Catalog Manager can use Presentation Manager® (PM) interface display messages that prompt the user for diskettes when necessary.

You can perform this linking using one of these methods:

– The following link statement:

  ilink /NOFREE /PMTYPE:vio /NOI *filename***.obj,,,dgwapi.lib,,**

– A module definition file. Specify an *apptype* of **WINDOWAPI** in your NAME statement.

Examples

Figure 66 shows the C language code required to invoke the FLGExport API call. This sample code exports three Information Catalog Manager objects. All three objects are Grouping objects:

- The first object, all the objects it contains, and its Contacts objects are exported.
- The second object, all the objects it contains, and attached Comments objects are exported.
- The third object is exported without exporting objects it contains.

```
APIRET         rc;     // Declare reason code
UCHAR   pszTagFileID[FLG_TAG_FILE_ID_MAXLEN + 1]; // Tag file id
UCHAR   pszLogFileID[FLG_LOG_FILE_ID_MAXLEN + 1]; // Log file id
UCHAR   pszIcoPath[FLG_ICON_PATH_MAXLEN + 1];  // icon files path
PFLGHEADERAREA  pListStruct;  // pointer to the input structure
FLGEXTCODE     ExtCode=0;   // declare an extended code for API
  .
  .  /* set values for Tag file/ Log file/ Icon path   */
  .  /* create object list             */
  .

rc  = FLGExport (pszTagFileID,
                 pszLogFileID,
                 pszIcoPath,
                 pListStruct,          // Pass input structure
                 &ExtCode);          // Pass pointer to extended code
```

Figure 66. Sample C language call to FLGExport

Figure 67 on page 119 shows the input structure for the FLGExport call.

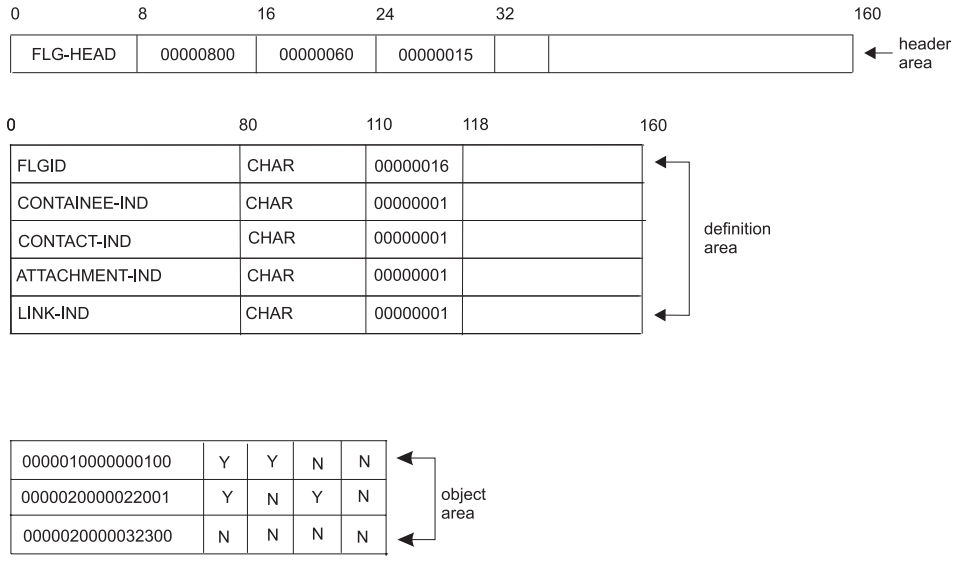

Figure 67. Sample input structure for FLGExport

FLGFoundIn

Retrieves a list of object instances or object types in which a specified instance is found. FLGFoundIn can retrieve any of the following:

- Grouping object instances that contain the specified object instance
- Object instances for which the specified object instance is a Contact
- Object instances to which the specified object instance is attached as a Comments object
- Object types with which the specified Programs object instance is associated

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGFoundIn( PSZ              pszFLGID,
                               FLGOPTIONS       Options,
                               PFLGHEADERAREA * ppListStruct,
                               PFLGEXTCODE      pExtCode );
```

Parameters

pszFLGID (PSZ) — input

Points to the 16-character object instance ID (FLGID) of the object instance for which a list of parents will be retrieved.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

The FLGID you specify depends on what you want to list:

Attachments
FLGID of an Attachment category object instance (retrieves object instances to which the specified object instance is attached as a Comments object).

Contacts
FLGID of a Contact category object instance (retrieves object instances for which the specified object instance is a Contact).

Containees
FLGID of an Elemental or Grouping category object instance (retrieves Grouping object instances that contain the specified object instance)

Programs
FLGID of a Program category object instance (retrieves object types with which the specified Programs object instance is associated)

Options (FLGOPTIONS) — input
Choose one of the following options:

FLG_LIST_ATTACHMENT
Retrieves object instances to which the specified object instance is attached as a Comments object

FLG_LIST_CONTACT
Retrieves object instances for which the specified object instance is a Contact

FLG_LIST_CONTAIN
Retrieves Grouping object instances that contain the specified object instance

FLG_LIST_PROGRAM
Retrieves object types with which the specified Programs object instance is associated

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure containing a list of object instances or object types in which a specified instance is found. If there is no output structure, then the pointer to the output structure is set to NULL.

For each Contain, Contact, or Attachment relationship, the output structure contains the following information about the "found-in" object instances:
- FLGID (16 characters)
- Name (80 characters)

All instances are sorted by object type name first, then object instance name, in ascending order according to collating order of the underlying database management system.

For each Program association, the output structure contains the following information about the "found-in" object types:
- Object type ID (6 characters)
- 80-character external name of object type (EXTERNAL NAME OF OBJ TYPE)

All object types are sorted by the 80-character external name of object type (EXTERNAL NAME OF OBJ TYPE) in ascending order according to collating order of the underlying database management system.

The maximum number of object instances or object types that can be returned by FLGFoundInis 5000.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Freeing memory allocated for an output structure

If FLGFoundIn returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

This sample code retrieves a list of object instances in which the specified Contact object is found. Figure 68 shows the C language code required to issue the FLGFoundIn call.

```
APIRET            rc;                // reason code from FLGFoundIn
UCHAR             pszInstID[FLG_ID_LEN + 1];
FLGOPTIONS        Option=0;          // association type
PFLGHEADERAREA  * ppReturnObjList;   // pointer to output structure ptr
FLGEXTCODE        xc=0;              // extended code
  .
  .  /* provide values for input parameters */
  .
Option = Option | FLG_LIST_CONTACT;
rc = FLGFoundIn (pszInstID,
                 Option,
                 ppReturnObjList,
                 &xc);
```

Figure 68. Sample C language call to FLGFoundIn

Figure 69 on page 123 shows the output structure for the FLGFoundIn call.

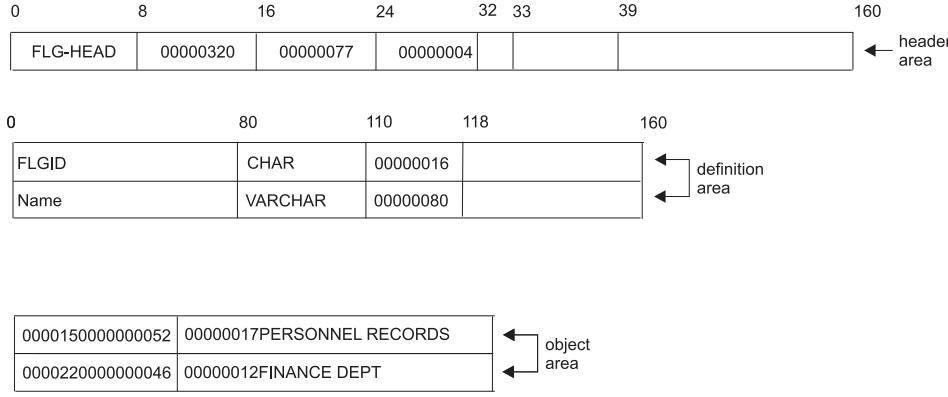

Figure 69. Sample output structure for FLGFoundIn

This sample code retrieves a list of object types handled by the specified Programs object instance. Figure 70 shows the C language code required to issue the FLGFoundIn call.

```
APIRET            rc;                // reason code from FLGFoundIn
UCHAR             pszInstID[FLG_ID_LEN + 1];
FLGOPTIONS        Option=0;          // association type
PFLGHEADERAREA  * ppReturnObjList;   // pointer to output structure ptr
FLGEXTCODE        xc=0;              // extended code
  .
  .  /* provide values for input parameters */
  .
Option = Option | FLG_LIST_PROGRAM;
rc = FLGFoundIn (pszInstID,
                 Option,
                 ppReturnObjList,
                 &xc);
```

Figure 70. Sample C language call to FLGFoundIn

Figure 71 on page 124 shows the output structure for the FLGFoundIn call.

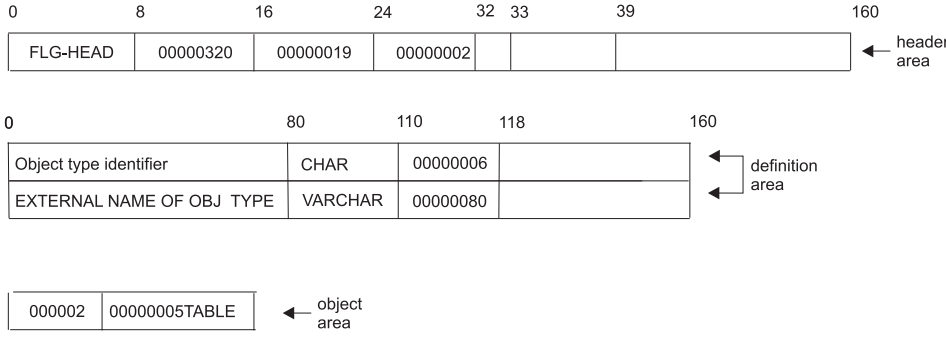

Figure 71. Sample output structure for FLGFoundIn

FLGFreeMem

Frees the memory allocated to an output structure created by an Information Catalog Manager API call; for example FLGListObjTypes or FLGNavigate.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY    FLGFreeMem( PFLGHEADERAREA   pFLGOutputStruct,
                                PFLGEXTCODE      pExtCode );
```

Parameters

pFLGOutputStruct (PFLGHEADERAREA) — input
Points to the Information Catalog Manager output structure to be deallocated.

When you issue an API call that creates an output structure, you need to save the value of the pointer to the output structure that is generated by the Information Catalog Manager and stored at the address indicated by the PFLGHEADERAREA data type so that you can pass this pointer as a parameter to FLGFreeMem to free the allocated memory.

FLGFreeMem works only with output structures produced by the Information Catalog Manager API calls.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Examples

Figure 72 on page 126 shows the C language code required to invoke the FLGFreeMem API call. This sample code frees an Information Catalog Manager output structure in memory.

```
PFLGHEADERAREA pFLGOutputStruct; // pointer to the FLG output structure
APIRET         rc;               // reason code
FLGEXTCODE     ExtCode = 0;           // Extended code

rc = FLGFreeMem ( pFLGOutputStruct, &ExtCode );
```

Figure 72. Sample C language call to FLGFreeMem

FLGGetInst

Retrieves a single object instance for a specified object type.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY     FLGGetInst( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppObjInstStruct,
                                 PFLGEXTCODE      pExtCode );
```

Parameters

pszFLGID (PSZ) — input
Points to the 16-character, system-generated unique identifier of the object instance to be retrieved.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

ppObjInstStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure. This pointer is set to NULL if FLGGetInst fails.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGGetInst produces an output structure containing the property specifications and values of the requested object instance, as shown in Figure 73 on page 128.

The object area of the output structure contains the values of the properties of the requested object instance.

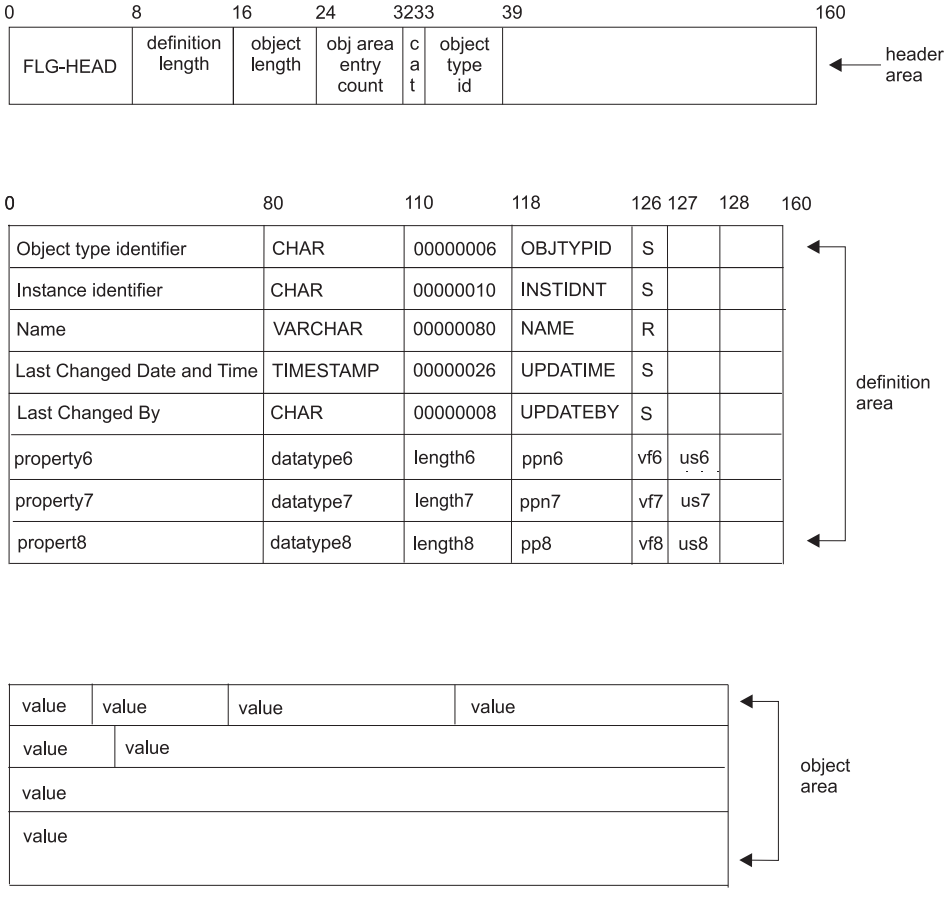

Figure 73. FLGGetInst output structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API output structure" on page 51.

Usage

Prerequisites

The value in the pszFLGID input parameter must refer to an existing object instance.

Freeing memory allocated for an output structure

If FLGGetInst returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Controlling updates to your information catalog

FLGGetInst commits changes to the database. Your program should issue FLGCommit or FLGRollback before issuing FLGGetInst to ensure that the Information Catalog Manager does not also commit unexpected changes that occurred before the FLGGetInst call.

Examples

Figure 74 shows the C language code required to invoke the FLGGetInst API call. This sample code retrieves information about the Quality Group object instance.

```
APIRET         rc;                       // Declare reason code
UCHAR          pszFLGID[FLG_ID_LEN+1];   // Unique ID for "Quality Group"
PFLGHEADERAREA * ppObjInstStruct;        // Pointer to the output structure
FLGEXTCODE     ExtCode = 0;                   // Declare extended code
  .
  .  /* Retrieving an object Instance  */
  .

strcpy (pszFLGID,"0000330000001234");
rc = FLGGetInst  (pszFLGID,        // Instance ID
                  ppObjInstStruct, // Structure pointer where output will be returned
                  &ExtCode);            // Pass pointer to extended code
```

Figure 74. Sample C language call to FLGGetInst

Figure 75 on page 130 shows the output structure that contains the property and value information for the object instance.

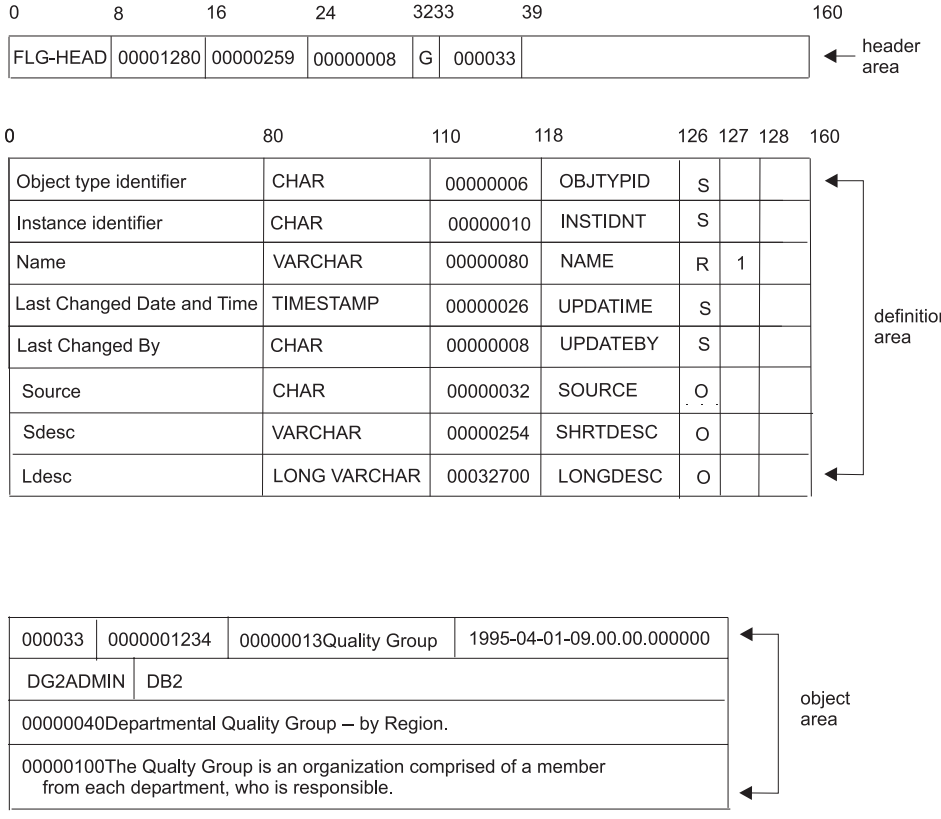

Figure 75. Sample output structure for FLGGetInst

FLGGetReg

Retrieves registration information from the information catalog for the specified object type.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGGetReg( PSZ              pszObjTypeID,
                              PSZ              pszIconFileID,
                              PFLGHEADERAREA * ppObjRegStruct,
                              PFLGEXTCODE      pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character, system-generated unique identifier (object type ID) of the object type for which you are retrieving the registration.

pszIconFileID (PSZ) — input/output
As input, points to the file path and name of the file in which you want to return the OS/2 icon for the registered object type. If this parameter is NULL, the Information Catalog Manager does not retrieve the icon for the registered object type.

As output, points to the file path and name of the file where the Information Catalog Manager stored the OS/2 icon for the registered object type. This pointer is set to NULL if there is no icon associated with the object type registration.

ppObjRegStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure.

The output structure contains the property specifications and values of the requested object type registration information. The pointer is set to NULL if FLGGetReg fails.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGGetReg produces an output structure containing the property specifications and values of the requested object type registration, as shown in Figure 76.

The object area of the output structure contains the values of the registration properties for the requested object type.

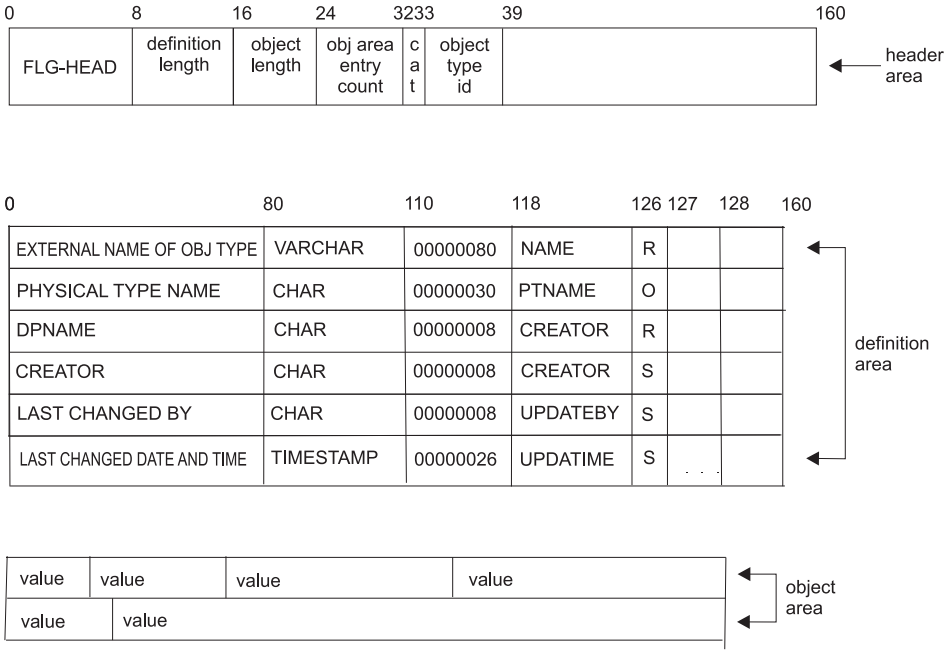

Figure 76. FLGGetReg output structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API output structure" on page 51.

Usage

Restrictions

You can only retrieve an OS/2 icon using FLGGetReg. To retrieve a Windows icon, use FLGManageIcons (see "FLGManageIcons" on page 182).

Prerequisites

The value in the pszObjTypeID parameter must refer to an existing object type ID registered in the information catalog.

Freeing memory allocated for an output structure

If FLGGetReg returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see

"FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 77 shows the C language code required to issue the FLGGetReg API call. This sample code retrieves information about the registration for the MYIMAGE object type from the information catalog.

```
APIRET         rc;                    // Declare reason code
UCHAR          pszObjTypeID[FLG_OBJTYPID_LEN+1]; // Unique ID for MYIMAGE ObjType
UCHAR          pszIconFileID[FLG_ICON_FILE_ID_MAXLEN+1]; // Path/File name for ICON
PFLGHEADERAREA * ppObjRegStruct;      // Ptr to pointer to the output structure
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  .  /* Retrieving an object Type Registration Instance  */
  .

strcpy (pszObjTypeID,"000044");
strcpy (pszIconFileID,"Y:\\FLGICON2.ICO");

rc = FLGGetReg    (pszObjTypeID, // id of the object type
                   pszIconFileID,// Path/File name of file to contain ICON
                   ppObjRegStruct,// Structure pointer where out  put will be returne
                   &ExtCode);          // Pass pointer to extended code
```

Figure 77. Sample C language call to FLGGetReg

Figure 78 on page 134 shows the output structure that contains the property and value information for the registration of the MYIMAGE object type.

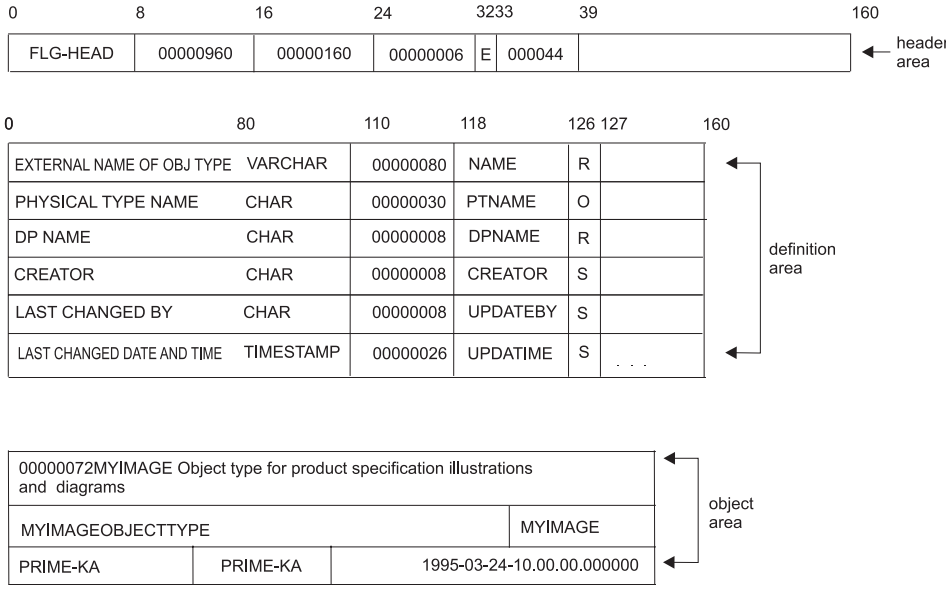

Figure 78. Sample output structure for FLGGetReg

In this example, bytes 33 through 38 of the header record contain the object type ID (000044) of the object type for which registration information has been retrieved. It matches the object type ID specified as input in the pszObjTypeID parameter.

FLGGetType

Retrieves the definition of all properties of an object type.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY    FLGGetType( PSZ              pszObjTypeID,
                                PFLGHEADERAREA * ppObjTypeStruct,
                                PFLGEXTCODE      pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character, system-generated unique identifier (object type ID) that was returned when the object type was registered. You can also retrieve this object type ID using either the FLGConvertID or FLGListObjTypes API call.

ppObjTypeStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure. The pointer is set to NULL if the FLGGetType fails.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGGetType produces an output structure containing the property specifications of the requested object type, as shown in Figure 79 on page 136.

The definition area of the output structure contains the properties of the requested object type in the order in which they were specified when the object type was created.

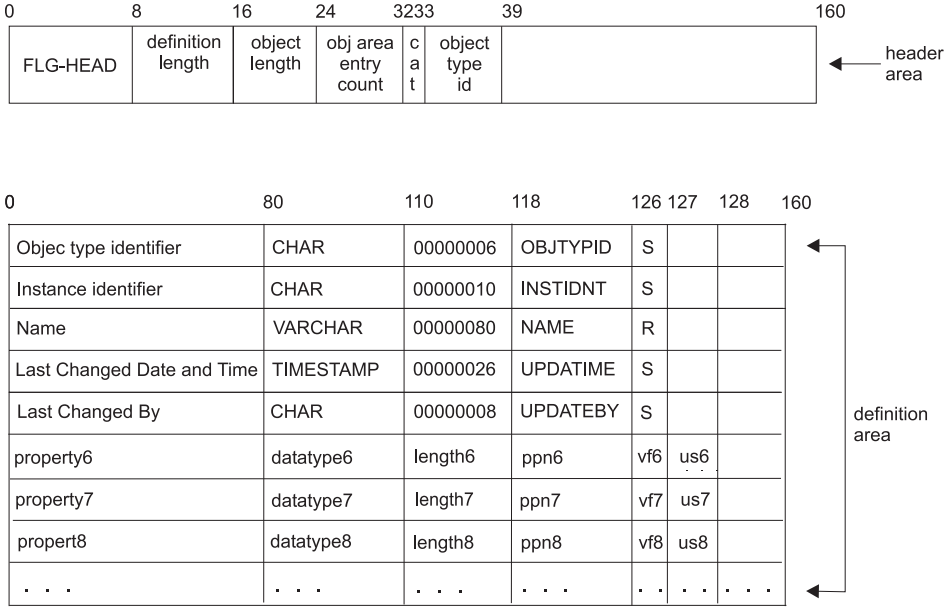

Figure 79. FLGGetType output structure

For an explanation of the meanings of the byte offsets, see "Chapter 4. The Information Catalog Manager input and output structures" on page 31.

Usage

Prerequisites

The value in the pszObjTypeID parameter must refer to an existing object type ID registered in the information catalog.

Freeing memory allocated for an output structure

If FLGGetType returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 80 on page 137 shows the C language code required to issue the FLGGetType API call. This sample code retrieves information about the properties of the MYIMAGE object type from the information catalog.

```
APIRET         rc;                    // Declare reason code
UCHAR          pszObjTypeID[FLG_OBJTYPID_LEN + 1]; // Set to ID of MYIMAGE (000044)
PFLGHEADERAREA * ppObjTypeStruct;     // Pointer to the output structure
FLGEXTCODE     ExtCode=0;                  // Declare extended code

  .
  .  /* retrieving a user-defined object type - MYIMAGE  */
strcpy (pszObjTypeID,"000044");

  .

rc = FLGGetType (pszObjTypeID,
                 ppObjTypeStruct,
                  &ExtCode);               // Pass pointer to extended code
```

Figure 80. Sample C language call to FLGGetType

Figure 81 shows the output structure that contains the property information for the MYIMAGE object type.

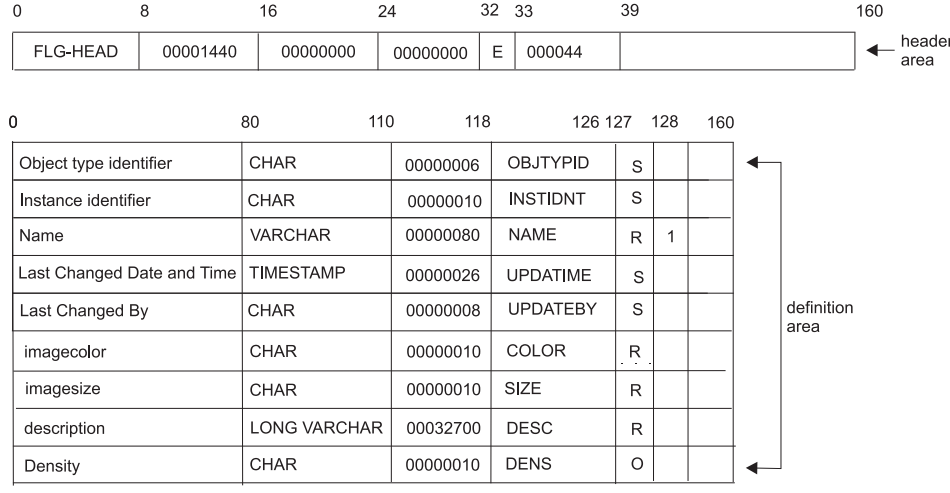

Figure 81. Sample output structure for FLGGetType

FLGImport

Imports metadata from a flat file in tag language format into the Information Catalog Manager.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGImport( PSZ             pszTagFileID,
                              PSZ             pszLogFileID,
                              PSZ             pszIcoPath,
                              FLGRESTARTOPTION RestartOpt,
                              PFLGEXTCODE     pExtCode );
```

Parameters

pszTagFileID (PSZ) — input
Identifies the tag language file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a FAT or HPFS file. This drive can be a removable drive. The file name and extension, excluding the drive and directories, cannot exceed 240 characters. If you type only the file name, the Information Catalog Manager assumes that the tag language file is on the drive and path pointed to by the DGWPATH environment variable.

The file identified by pszTagFileID contains the Information Catalog Manager objects and related metadata to be imported.

pszLogFileID (PSZ) — input
Specifies the location and name of the log file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a FAT or HPFS file. The drive cannot be a removable drive. If you specify only a file name, the Information Catalog Manager places the log file on the drive and path pointed to by the DGWPATH environment variable.

If the log file specified in this parameter does not exist, a new file is created. If the log file specified in this parameter already exists, then the Information Catalog Manager appends to it.

The file identified by pszLogFileID contains logging information as well as warnings and errors detected during processing of the FLGImport API call.

pszIcoPath (PSZ) — input
Specifies the location of the OS/2 and Windows icon files. This parameter

contains the drive and directories, and must be valid for a FAT or HPFS file on a nonremovable drive. The maximum length for the icon path is 246 characters.

This parameter is optional. If you do not specify this parameter, icon files are not imported, even when the tag language file contains instructions to import icons associated with object types.

When specified, the import function searches this path for any icon files referenced within the tag language file identified by pszTagFileID. If the tag language file indicates that icons are to be associated with an object type, and the icons do not reside in the icon path, a warning is recorded in the log file.

RestartOpt (FLGRESTARTOPTION) — input
Specifies whether the Information Catalog Manager processes the input tag language file from the beginning or from a checkpoint. Valid values are:

B Beginning

The tag language file is processed from the beginning, even if the same tag language file was already specified at a previous time and only partially processed because of run-time errors.

C Checkpoint

The same tag language file was processed, but only partially. The system saved the checkpoint label information where execution is to resume for this file. In this case, the tag language file is searched for the saved checkpoint label and, if a match is found, importing resumes from that point. If a match is not found, then the FLGImport API call fails.

If C is specified for the RestartOpt, but the tag language file was not previously processed, then the Information Catalog Manager processes the tag language file from the beginning.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Debugging import errors

The Information Catalog Manager creates a *log file* and an *echo file* when importing a tag language file.

The log file records what happens during the import process. It includes the times and dates when the import process started and stopped. It also includes any warning or error messages for problems that occur during the process. The log file is identified by the pszLogFileID parameter.

The echo file lists the tags that have been processed by the Information Catalog Manager. The echo file has the same name as the import tag language file, and is stored in the same directory and path as the log file, but has the `.ech` file extension.

You can use the echo file and log file to find the tag that is causing the import error. The last one or two tags of an echo file tell you which tag in your tag language file caused the import process to stop.

Importing a delete history tag file

To protect against erroneous deletions in other information catalogs, you should examine the contents of a delete history tag file before importing it to any other information catalog, especially if you have deleted Grouping object instances, or object types.

Linking your VisualAge® C++ program when it imports from diskettes

If your C language program issues an FLGImport call that imports the Information Catalog Manager information from diskettes, link your program with an application type of WINDOWAPI so that the Information Catalog Manager can use the PM interface to display messages that prompt the user for diskettes when necessary.

You can perform this linking using one of these methods:

– The following link statement:

  ilink /NOFREE /PMTYPE:vio /NOI *filename***.obj,,,dgwapi.lib,,**

– A module definition file. Specify an *apptype* of **WINDOWAPI** in your NAME statement.

Committing changes before using FLGImport

The Information Catalog Manager rolls back the database when FLGImport encounters errors. Your program should issue FLGCommit before issuing FLGImport to ensure that the Information Catalog Manager does not also roll back uncommitted changes that occurred before the FLGImport call.

Examples

The sample code in Figure 82 on page 141 imports a tag language file named TAGFILE1.TAG. The Information Catalog Manager logs the processing information in TAGFILE1.LOG.

```
APIRET           rc;                                  // Declare reason code
UCHAR            pszTagFileID[FLG_TAG_FILE_ID_MAXLEN+1];  // ID for Tag Language file
UCHAR            pszLogFileID[FLG_LOG_FILE_ID_MAXLEN+1];  // ID for Log file
UCHAR            pszIconPath[FLG_ICON_PATH_MAXLEN+1];  // Path for Icon files
FLGRESTARTOPTION RestartOpt;                          // Restart option
FLGEXTCODE       ExtCode=0;            // Returned extended code
  .
  .  /* Importing the Tag Language file TAGFILE1.TAG              */
  .

strcpy (pszTagFileID,"c:\\DGdata\\TAGFILE1.TAG");
strcpy (pszLogFileID,"c:\\DGdata\\TAGFILE1.LOG");
strcpy (pszIconPath,"c:\\DGdata");
RestartOpt = FLG_RESTART_BEGIN;

rc = FLGImport (pszTagFileID,
                pszLogFileID,
                pszIconPath,
                RestartOpt,
                &ExtCode);        // Pass extended code by reference
```

Figure 82. Sample C language call to FLGImport

FLGInit

Initializes the Information Catalog Manager API DLL for use, connects the application to the database, and retrieves environmental information that you can use with other API calls.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGInit( PSZ              pszUserID,
                            PSZ              pszPassword,
                            PSZ              pszDatabaseName,
                            FLGADMIN         Admin,
                            PFLGHEADERAREA * ppListStruct,
                            PFLGEXTCODE      pExtCode );
```

Parameters

pszUserID (PSZ) — input
Points to a null-terminated string containing the user ID for the information catalog database logon.

pszPassword (PSZ) — input
Points to a null-terminated string containing the user's password.

pszDatabaseName (PSZ) — input
Points to a null-terminated string containing the database alias for the information catalog.

admin (FLGADMIN) — input
Indicates the user option desired.
FLG_YES
Log on as an administrator.
FLG_NO
The default. Log on as an user.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure. For the format of the output structure, see "The Information Catalog Manager API output structure" on page 51.

If there is no output structure, the pointer to the output structure is set to NULL, and the Information Catalog Manager returns an error condition with a reason code.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See

"Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGInit produces an output structure containing information about the Information Catalog Manager environment, as shown in Figure 83 on page 144.

The object area of the output structure contains the required registration and object properties in the user's national language. The object area also contains values that provide information about the user's Information Catalog Manager environment.

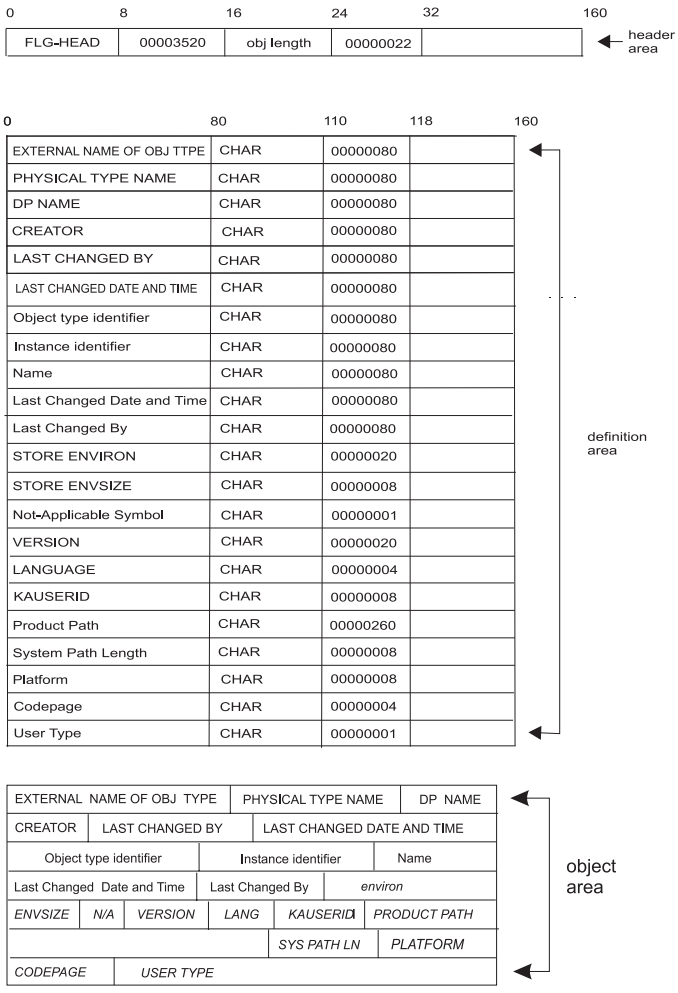

Figure 83. FLGInit output structure

Usage

The output structure returns the 80-byte property names required for all object type registrations, object types, and objects.

If you are using a non-English version of the Information Catalog Manager, the values in the object area for each of these properties are translated. You need to save these translated values so that you can use them with FLGCreateReg and FLGCreateType.

Freeing memory allocated for an output structure

If FLGInit returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see

"FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Table 17 shows the required properties that are returned by FLGInit.

Table 17. Required property names returned by FLGInit

Property name	Description
EXTERNAL NAME OF OBJ TYPE	First registration property in any object type registration
PHYSICAL TYPE NAME	Second registration property in any object type registration
DP NAME	Third registration property in any object type registration
CREATOR	Fourth registration property in any object type registration
LAST CHANGED BY	Fifth registration property in any object type registration
LAST CHANGED DATE AND TIME	Sixth registration property in any object type registration
Object type identifier	First required property on any object type
Instance identifier	Second required property on any object type
Name	Third required property on any object type
Last Changed Date and Time	Fourth required property on any object type
Last Changed By	Fifth required property on any object type

This output structure also returns environmental values. Save these values to use with other API calls.

Table 18. Environmental values returned by FLGInit

Property name	Description
STORE ENVIRON	Database product name with the release number in V*x*R*x*M*x* format. For example: **DB2/NT V07R01M0** DB2 UDB for Windows NT product
STORE ENVSIZE	Value indicating the maximum length of PTNAME for the information catalog in this environment.
Not-applicable symbol	1-character default token of the Information Catalog Manager environment to represent an unspecified data field. This value was set during installation.

Table 18. Environmental values returned by FLGInit (continued)

Property name	Description
VERSION	20-character indicator of the version of the Information Catalog Manager.
LANGUAGE	3-character national language code; for example, ENU indicates English. Valid values are: **CHS** Simplified Chinese **CHT** Traditional Chinese **DAN** Danish **DEU** German **ENU** US English **ESP** Spanish **FIN** Finnish **FRA** French **ITA** Italian **JPN** Japanese **KOR** Korean **NLB** Belgian French **NOR** Norwegian **PTB** Brazilian Portuguese **SVE** Swedish
KAUSERID	8-character user ID for the administrator currently logged on.
Product Path	260-character full working path for the Information Catalog Manager.
System Path Length	8-character value for the maximum path length for the system.
Code page	4-character code page identifier
User Type	1-character identifier, set to: **A** Logged-on user ID is the primary administrator **B** Logged-on user ID is the backup administrator **D** Logged-on user ID is a user with authority to perform object management tasks **W** Logged-on user ID is a user

Examples

Figure 84 on page 147 shows the C language code required to invoke the FLGInit API call. This sample code initializes the Information Catalog Manager API DLL so that information applications can issue calls to the Information Catalog Manager API.

```
UCHAR          pszUserID[FLG_USERID_LEN + 1];
UCHAR          pszPassword[FLG_PASSWORD_LEN + 1];
UCHAR          pszDatabaseName[FLG_DATABASENAME_LEN + 1];
FLGADMIN       admin = FLG_YES;
APIRET         rc;               // reason code
PFLGHEADERAREA * ppListStruct;    // pointer to output structure pointer
FLGEXTCODE     ExtCode = 0;           // Extended code

.
.  // IA specific code
.
strcpy( pszUserID, "LAUTZ" );
strcpy( pszPassword, "MYPASSWD");
strcpy( pszDatabaseName, "CATALOG");

rc = FLGInit (pszUserName,
              pszPassword,
              pszDatabaseName,
              admin,
              ppListStruct,
              &ExtCode );

.  // Issue FLGFreeMem to release the output structure created by FLGInit
.  // Calls to the FLG API
.  // When complete, call
.  // FLGTerm()
```

Figure 84. Sample C language call to FLGInit

Figure 85 on page 148 shows the output structure.

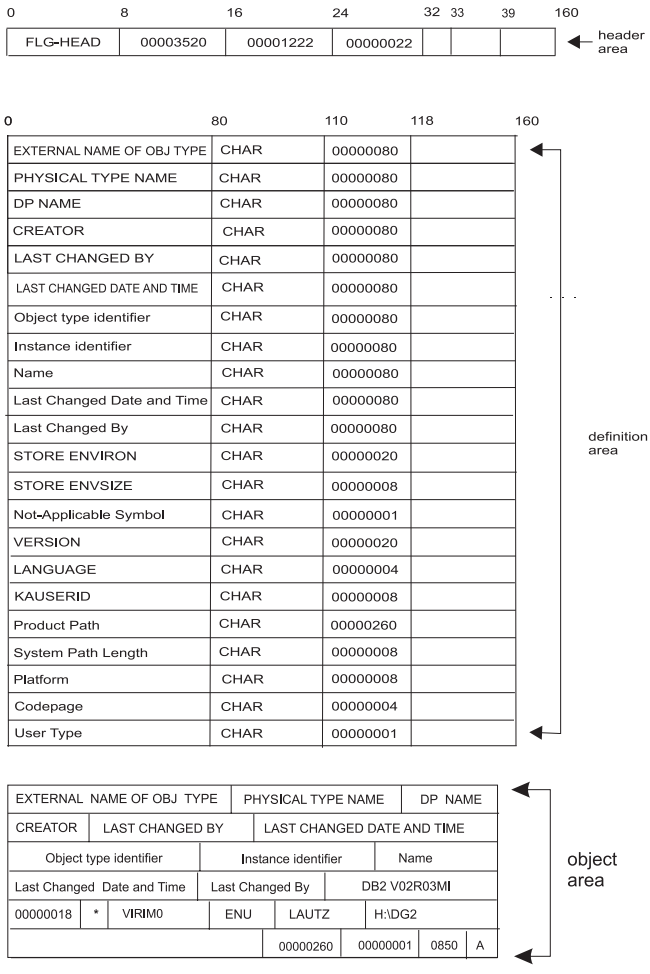

Figure 85. Sample output structure for FLGInit

FLGListAnchors

Retrieves a list of all anchor instances for the Grouping category. *Anchors* are Grouping category objects that have containees, but are not contained by other objects.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGListAnchors( PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the anchors. When there is no output structure, the pointer to the structure is set to NULL.

The output structure contains the following information for each anchor object instance:
- FLGID (16 characters)
- Name (80 characters)

All instances are sorted according to the collating sequence of the database used for your information catalog, first by object type name, then by Name.

The maximum number of object instances that can be returned by FLGListAnchors is 1600.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGListAnchors produces an output structure containing a list of anchors, as shown in Figure 86 on page 150.

The object area of the output structure contains a list of anchor object instances, identified by the value of the FLGID and the external name for each object instance.

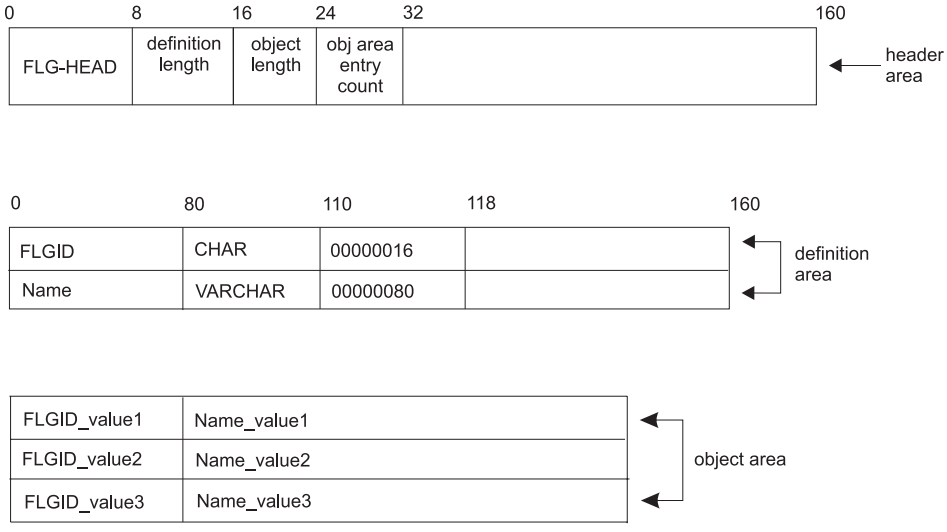

Figure 86. FLGListAnchors output structure

Usage

Freeing memory allocated for an output structure

If FLGListAnchors returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 87 on page 151 shows the C language code required to invoke the FLGListAnchors API call. This sample code retrieves a list of the anchors in your information catalog.

```
APIRET          rc;              // reason code from FLGListAnchors
PFLGHEADERAREA  * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE      ExtCode=0;           // Extended code
  .
  .
rc = FLGListAnchors (ppListStruct,   // address of output structure pointer
                     &ExtCode);
```

Figure 87. Sample C language call to FLGListAnchors

Figure 88 shows the output structure.

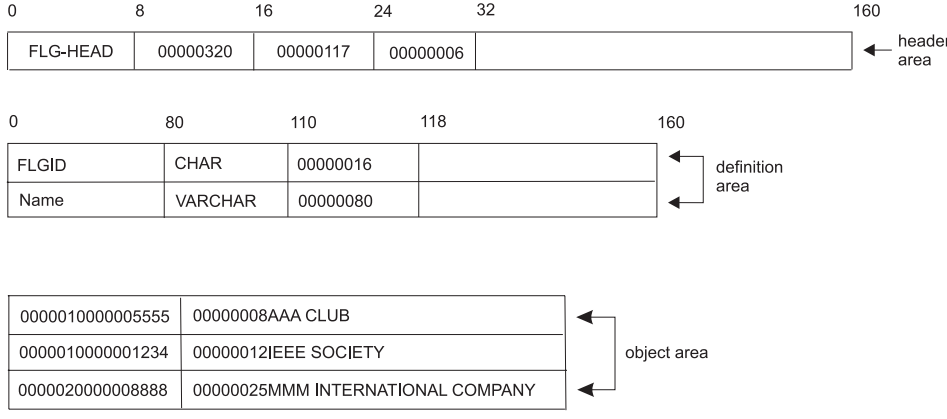

Figure 88. Sample output structure for FLGListAnchors

FLGListAssociates

Retrieves a list of associate instances for a specified object instance or object type. An associate can be any one of the following:

- Instances contained by a Grouping object instance
- Contact instances for an object instance
- Attachment instances for an object instance
- Instances linked with an object instances
- Program instances associated with an object type

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGListAssociates( PSZ              pszInBuffer,
                                      FLGOPTIONS       Options,
                                      PFLGHEADERAREA * ppListStruct,
                                      PFLGEXTCODE      pExtCode );
```

Parameters

pszInBuffer (PSZ) — input

Points to an input buffer containing either a 16-character, system-generated unique identifier of an object instance, or a 6-character, system-generated unique identifier of an object type, depending on what you are listing:

Attachments
Object instance ID (FLGID) of a non-Attachment category object instance

Comments
FLGID of a non-Comments type object instance

Contacts
FLGID of an Elemental or Grouping category object instance

Containees
FLGID of a Grouping category object instance

Links FLGID of an Elemental or Grouping category object instance

Programs
Object type ID of a non-Program category object type

Options (FLGOPTIONS) — input

Choose one of the following options:

FLG_LIST_ATTACHMENT
Retrieves object instances in an Attachment relationship with the specified instance.

FLG_LIST_COMMENTS
Retrieves Comments object instances attached to the specified instance. FLG_LIST_COMMENTS retrieves the same object instances as FLG_LIST_ATTACHMENT, but returns more information (Last Changed Date and Time, Creator) about each instance.

FLG_LIST_CONTACT
Retrieves Contact object instances associated with the specified instance.

FLG_LIST_CONTAIN
Retrieves object instances contained in the specified instance.

FLG_LIST_LINK
Retrieves object instances linked with the specified instance.

FLG_LIST_PROGRAM
Retrieves Programs object instances associated with the specified object type.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the associates. When there is no output structure, the pointer to the structure is set to NULL.

The output structure for each instance has the following information:

FLGID (16 characters)

Name (80 characters)

In addition, for FLG_LIST_CONTAIN, the output structure for each instance also has a flag (CHILDIND) indicating whether it is itself a container. For FLG_LIST_COMMENTS, the output structure for each instance also includes the following:

Last Changed Date and Time

Creator

All instances are sorted by object type name first, then object instance name, in ascending order according to collating order of the underlying database management system.

The maximum number of object instances that can be returned by FLGListAssociates is 5000.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Freeing memory allocated for an output structure

If FLGListAssociates returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

This sample code retrieves a list of the Programs object instances associated with the Grouping object type, MYREGION. Figure 89 shows the C language code required to issue the FLGListAssociates call.

```
APIRET           rc;               // reason code
UCHAR            pszObjTypeID[FLG_OBJTYPID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_PROGRAM;
rc = FLGListAssociates (pszObjTypeID,
                        Option,
                        ppReturnObjList,
                        &xc);
```

Figure 89. Sample C language call to FLGListAssociates

Figure 90 on page 155 shows the output structure for the FLGListAssociates call in Figure 89.

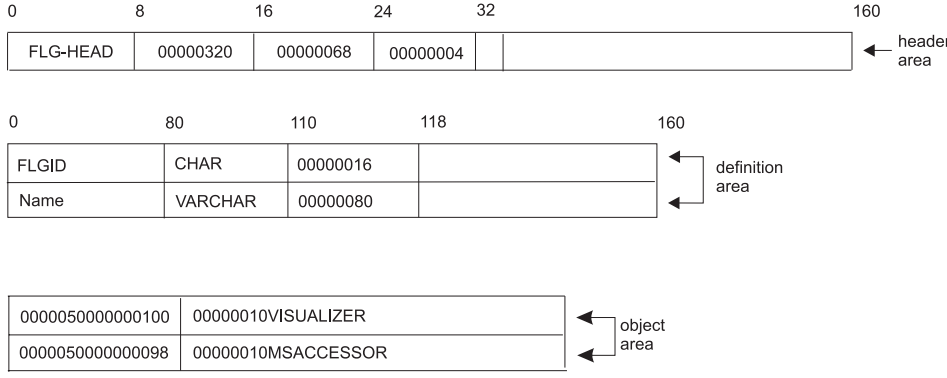

Figure 90. Sample output structure for FLGListAssociates

This sample code retrieves the object instances contained in the Grouping object, MYBGROUP. Figure 91 shows the C language code required to issue the FLGListAssociates call.

```
APIRET           rc;               // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_CONTAIN;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

Figure 91. Sample C language call to FLGListAssociates

Figure 92 on page 156 shows the output structure for the FLGListAssociates call in Figure 91.

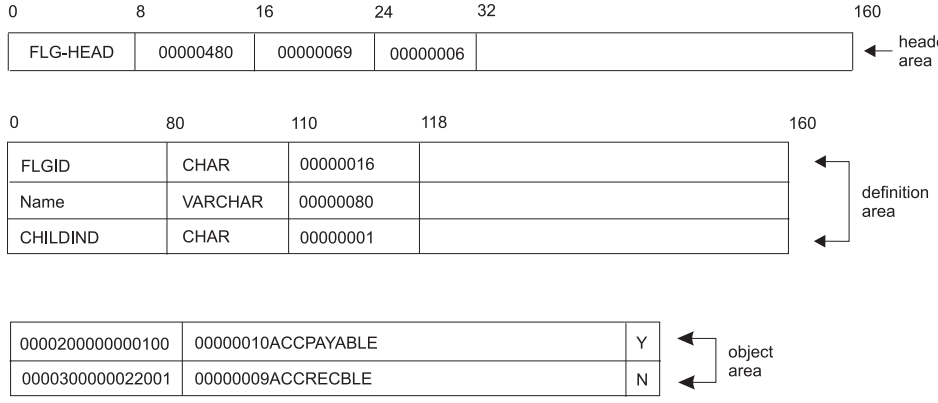

Figure 92. Sample output structure for FLGListAssociates

This sample code retrieves the Contact object instances for the Grouping object, MYBGROUP. Figure 93 shows the C language code required to issue the FLGListAssociates call.

```
APIRET           rc;               // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_CONTACT;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

Figure 93. Sample C language call to FLGListAssociates

Figure 94 on page 157 shows the output structure for the FLGListAssociates call in Figure 93.

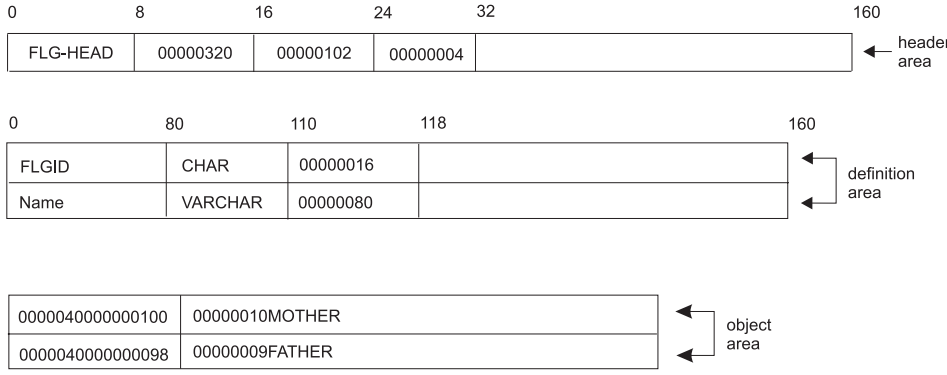

Figure 94. Sample output structure for FLGListAssociates

This sample code retrieves the Attachment category object instances for the Grouping object, MYBGROUP. Figure 95 shows the C language code required to issue the FLGListAssociates call.

```
APIRET           rc;               // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_ATTACHMENT;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

Figure 95. Sample C language call to FLGListAssociates

Figure 96 on page 158 shows the output structure for the FLGListAssociates call in Figure 95.

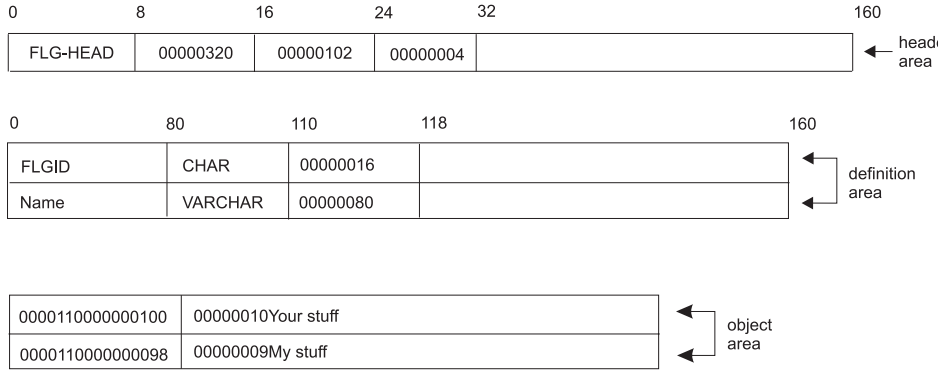

Figure 96. Sample output structure for FLGListAssociates

This sample code retrieves the Comments object instances attached to the Elemental object, MYREPORT. Figure 97 shows the C language code required to issue the FLGListAssociates call.

```
APIRET           rc;               // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_COMMENTS;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

Figure 97. Sample C language call to FLGListAssociates

Figure 98 on page 159 shows the output structure for the FLGListAssociates call in Figure 97.

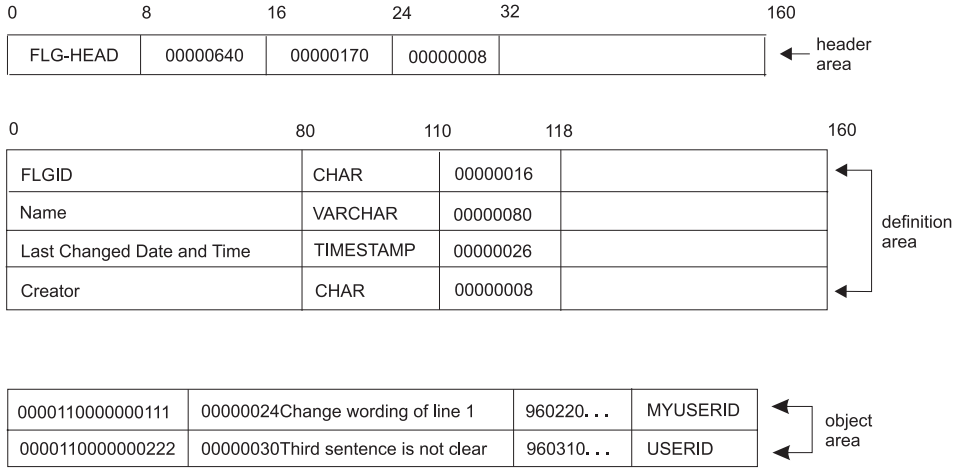

Figure 98. Sample output structure for FLGListAssociates

This sample code retrieves the object instances with which the Grouping object, MYBGROUP, is linked. Figure 99 shows the C language code required to issue the FLGListAssociates call.

```
APIRET           rc;               // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_LINK;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

Figure 99. Sample C language call to FLGListAssociates

Figure 100 on page 160 shows the output structure for the FLGListAssociates call in Figure 99.

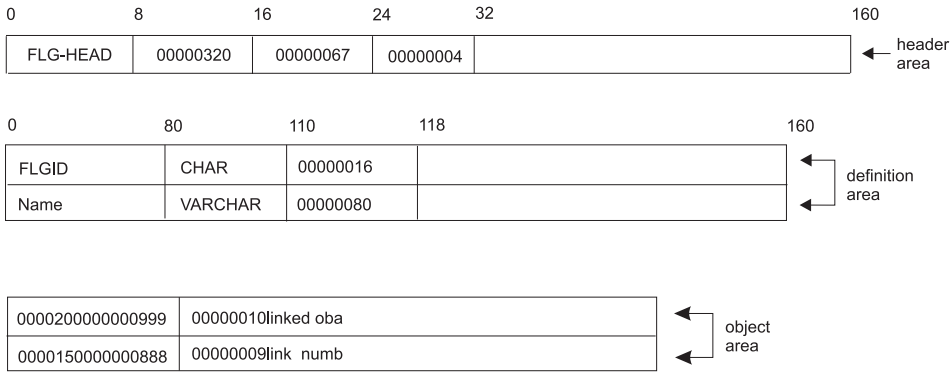

Figure 100. Sample output structure for FLGListAssociates

FLGListContacts

Retrieves a list of Contact objects for an Elemental or Grouping object.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY  FLGListContacts( PSZ              pszFLGID,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

pszFLGID (PSZ) — input
Points to the 16-character FLGID of the object instance for which Contacts will be retrieved.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the Contacts. When there is no output structure, the pointer to the structure is set to NULL.

This output structure contains the 16-character FLGID of each Contact object and its 80-character name.

Entries in the list are first sorted by object type name, then by the value of the Name property for each instance, according to the collating sequence used by the database management system used by your information catalog.

The maximum number of Contact object instances that can be returned by FLGListContacts is approximately 5000, depending on the storage available on your machine.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGListContacts produces an output structure containing a list of Contacts, as shown in Figure 101.

The object area of the output structure contains a list of Contact object instances associated with the specified object instance. These Contact objects are identified by the value of the FLGID and the external name for each object instance.

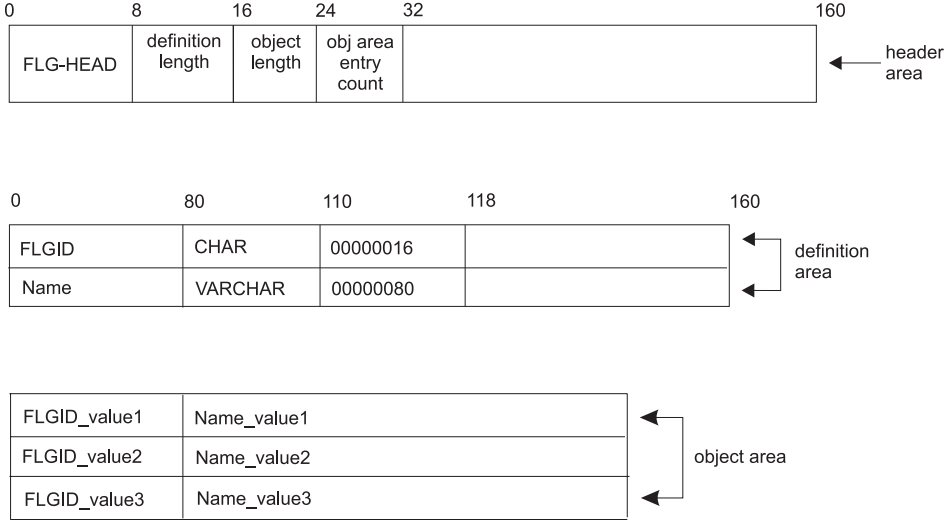

Figure 101. FLGListContacts output structure

Usage

Freeing memory allocated for an output structure

If FLGListContacts returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 102 on page 163 shows the C language code required to invoke the FLGListContacts API call. This sample code retrieves a list of the Contacts for Elemental object MYREPORT.

```
APIRET           rc;               // reason code from FLGListContacts
UCHAR            pszFLGID[FLG_ID_LEN + 1];
PFLGHEADERarea * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE       ExtCode=0;             // extended code
  .
  .  /* allocate storage for input parms               */
  .  /* set objid to FLGID of 'MYREPORT'              */
  .
rc = FLGListContacts (pszFLGID,
                      ppListStruct,  // address of output structure pointer
                      &ExtCode);
```

Figure 102. Sample C language call to FLGListContacts

Figure 103 shows the output structure for this API call.

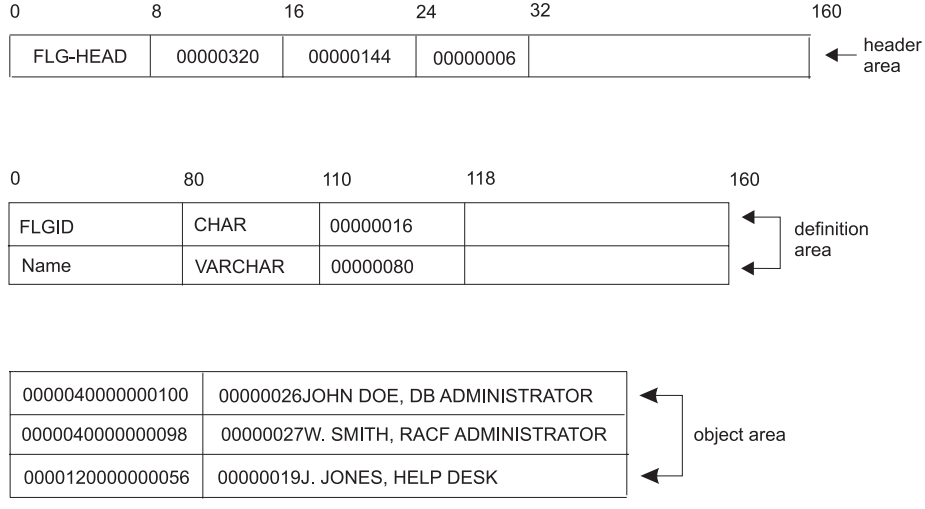

Figure 103. Sample output structure for FLGListContacts

FLGListObjTypes

Displays all object types currently registered and created in the information catalog database.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY  FLGListObjTypes( PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE     pExtCode );
```

Parameters

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the object types. When there is no output structure, the pointer to the structure is set to NULL.

Each entry has the following information:
- Object type ID
- Object type external name (80-byte)
- Object type short name (8-byte DP NAME)

Entries are sorted by 80-byte object type external name (EXTERNAL NAME OF OBJ TYPE); the actual order depends on the collating sequence used by the database management system used for the information catalog.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGListObjTypes produces an output structure containing a list of object types, as shown in Figure 104 on page 165.

The object area of the output structure contains a list of all the object types in the information catalog. These object types are identified by the values of the

object type ID, the object type external name, and the object type DP NAME (short name).

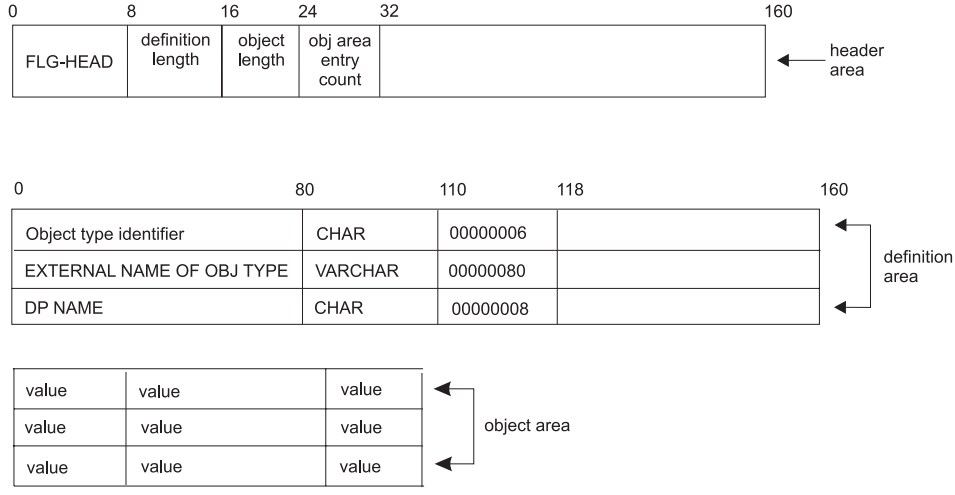

Figure 104. FLGListObjTypes output structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API output structure" on page 51.

Usage

Freeing memory allocated for an output structure

If FLGListObjTypes returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 105 on page 166 shows the C language code required to invoke the FLGListObjTypes API call. This sample code retrieves a list of all the object types in the information catalog.

```
PFLGHEADERAREA * ppListStruct;  // pointer to output structure pointer
APIRET          rc;            // reason code from FLGListObjTypes
FLGEXTCODE      ExtCode=0;          // extended code
  .
  .
  .
rc = FLGListObjTypes (ppListStruct,  // address of output structure pointer
                      &ExtCode );
```

Figure 105. Sample C language call to FLGListObjTypes

Figure 106 shows the output structure for this API call.

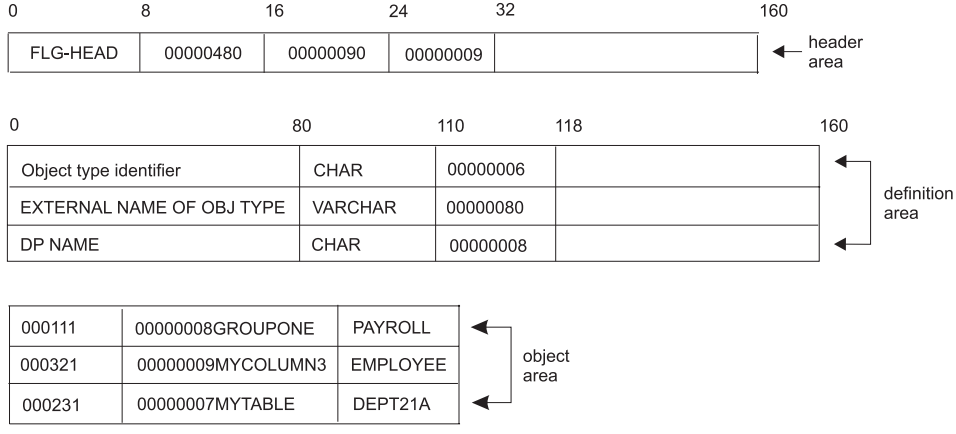

Figure 106. Sample output structure for FLGListObjTypes

FLGListOrphans

Retrieves a list of all orphan instances of the Attachment, Contact, or Program category. *Orphans* are Attachment, or Contact objects that are not associated with other object instances, or Program objects that are not associated with any object type.

You can use this list to clean up your information catalog by associating orphan object instances to other objects or by deleting orphan instances.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGListOrphans( PSZ              pszObjTypeID,
                                   FLGOPTIONS       Options,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character, system-generated unique identifier (object type ID) of an object type for which to retrieve a list of instances that exist, but are not currently associated with any object instances. The object type ID you specify depends on what you want to list:

Attachments
Attachment category object type ID

Comments
This parameter is ignored.

Contacts
Contact category object type ID

Programs
Program category object type ID

If pszObjTypeID is NULL, then the Information Catalog Manager returns orphans of all object types in the Attachment category (when FLG_LIST_ATTACHMENT is specified), or in the Contact category (when FLG_LIST_CONTACT is specified).

Options (FLGOPTIONS) — input
Choose one of the following options:

FLG_LIST_ATTACHMENT
Retrieves Attachment category object instances that are currently unattached.

FLG_LIST_COMMENTS
Retrieves Comments object instances that are currently unattached. FLG_LIST_COMMENTS retrieves the same object instances as FLG_LIST_ATTACHMENT, but returns more information (Last Changed Date and Time, Creator) about each instance

FLG_LIST_CONTACT
Retrieves Contact category object instances that are currently unattached.

FLG_LIST_PROGRAM
Retrieves Programs object instances that are not currently associated with any object type.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the orphans. When there is no output structure, the pointer to the structure is set to NULL.

The output structure for each instance has the following information:

FLGID (16 characters)

Name (80 characters)

In addition, for FLG_LIST_COMMENTS, the output structure for each instance also includes the following:

Last Changed Date and Time

Creator

All instances are sorted by object type name first, then object instance name, in ascending order according to collating order of the underlying database management system.

The maximum number of object instances that can be returned by FLGListOrphans is 1600.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Restrictions

If a user uses FLGListOrphans to list orphan Comments, FLGListOrphans only returns the Comments for which the user is also the creator.

Freeing memory allocated for an output structure

If FLGListOrphans returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

This sample code retrieves all orphan Program category object instances. Figure 107 shows the C language code required to issue the FLGListOrphans call.

```
APIRET           rc;               // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_PROGRAM;
rc = FLGListOrphans  (NULL,
                      Option,
                      ppReturnObjList,
                      &xc);
```

Figure 107. Sample C language call to FLGListOrphans

Figure 108 shows the output structure for the FLGListOrphans call in Figure 107.

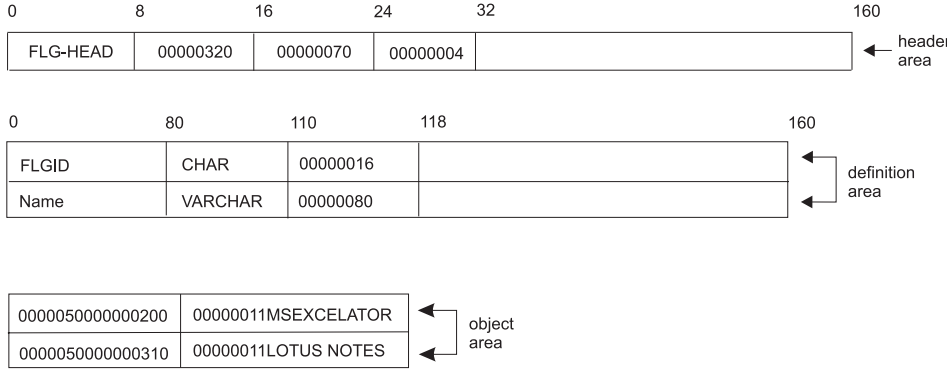

Figure 108. Sample output structure for FLGListOrphans

This sample code retrieves all orphan Contact category object instances. Figure 109 shows the C language code required to issue the FLGListOrphans call.

```
APIRET           rc;               // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_CONTACT;
rc = FLGListOrphans  (NULL,
                      Option,
                      ppReturnObjList,
                      &xc);
```

Figure 109. Sample C language call to FLGListOrphans

Figure 110 shows the output structure for the FLGListOrphans call in Figure 109.

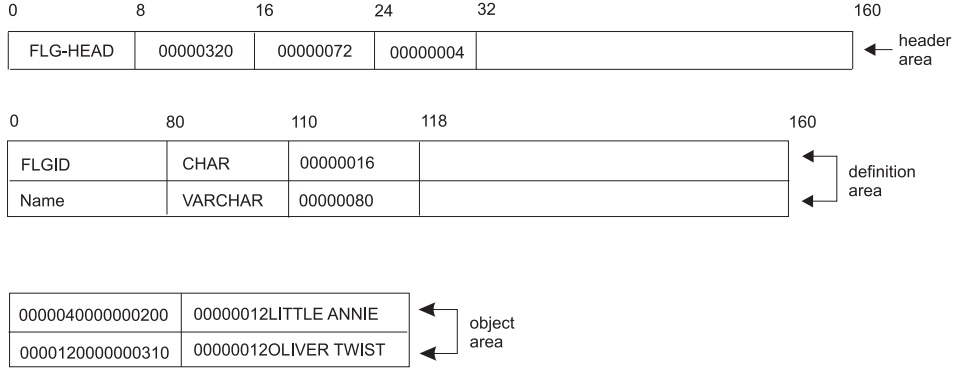

Figure 110. Sample output structure for FLGListOrphans

This sample code retrieves all orphan Attachment category object instances. Figure 111 on page 171 shows the C language code required to issue the FLGListOrphans call.

```
APIRET           rc;               // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_ATTACHMENT;
rc = FLGListOrphans  (NULL,
                      Option,
                      ppReturnObjList,
                      &xc);
```

Figure 111. Sample C language call to FLGListOrphans

Figure 112 shows the output structure for the FLGListOrphans call in Figure 111.

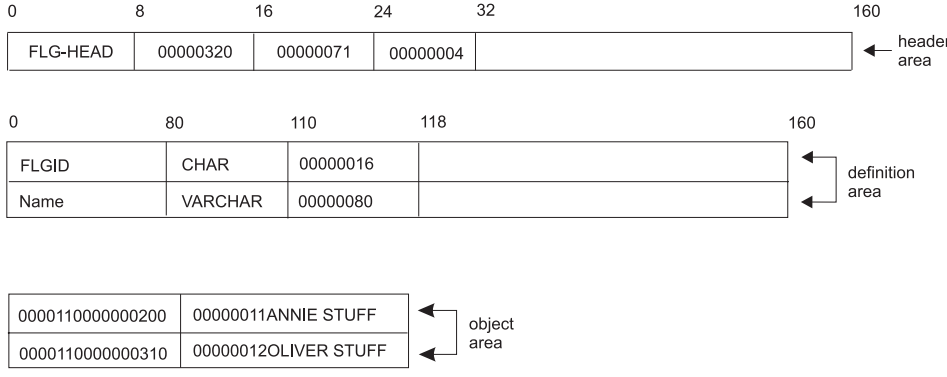

Figure 112. Sample output structure for FLGListOrphans

This sample code retrieves all orphan Attachment category object instances that are of the Comments object type. Figure 113 on page 172 shows the C language code required to issue the FLGListOrphans call.

```
APIRET           rc;               // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_COMMENTS;
rc = FLGListOrphans (NULL,
                     Option,
                     ppReturnObjList,
                     &xc);
```

Figure 113. Sample C language call to FLGListOrphans

Figure 114 shows the output structure for the FLGListOrphans call in Figure 113. This particular output structure has two additional property values.

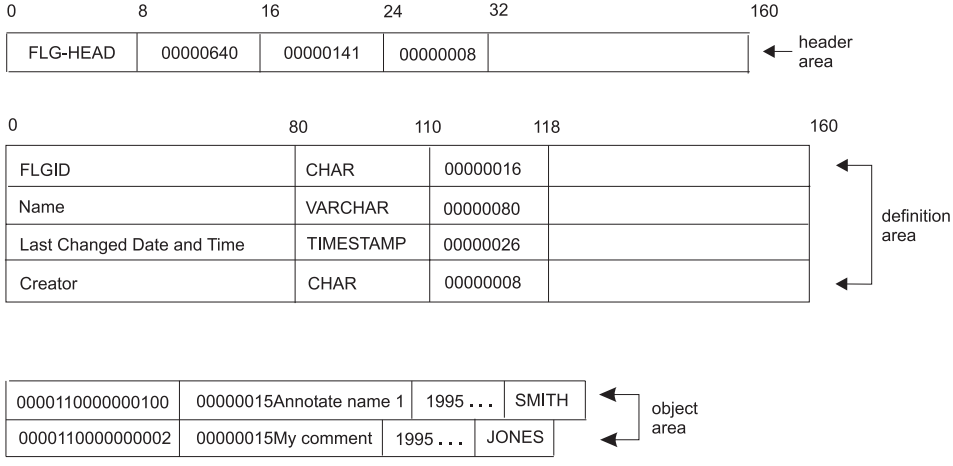

Figure 114. Sample output structure for FLGListOrphans

FLGListPrograms

Retrieves a list of Programs objects for a non-Program object type.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY  FLGListPrograms( PSZ              pszObjTypeID,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character, system-generated unique identifier (object type ID) of the object type for which to retrieve a list of associated Programs objects.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the Programs instances. When there is no output structure, the pointer to the structure is set to NULL.

This output structure contains the 16-character FLGID of a Programs object instance and its 80-character external name.

Entries in the list are sorted by the external name (value of the NAME property); the actual order of the list depends on the collating sequence used by the database management system used for your information catalog.

The maximum number of Programs object instances that can be returned by FLGListPrograms is approximately 5000, depending on the storage available on your machine.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGListPrograms produces an output structure containing a list of Programs objects, as shown in Figure 115 on page 174.

The object area of the output structure contains a list of all the Programs objects associated with the specified object type. These Programs objects are identified by the values of the FLGID and the external name of the object instance.

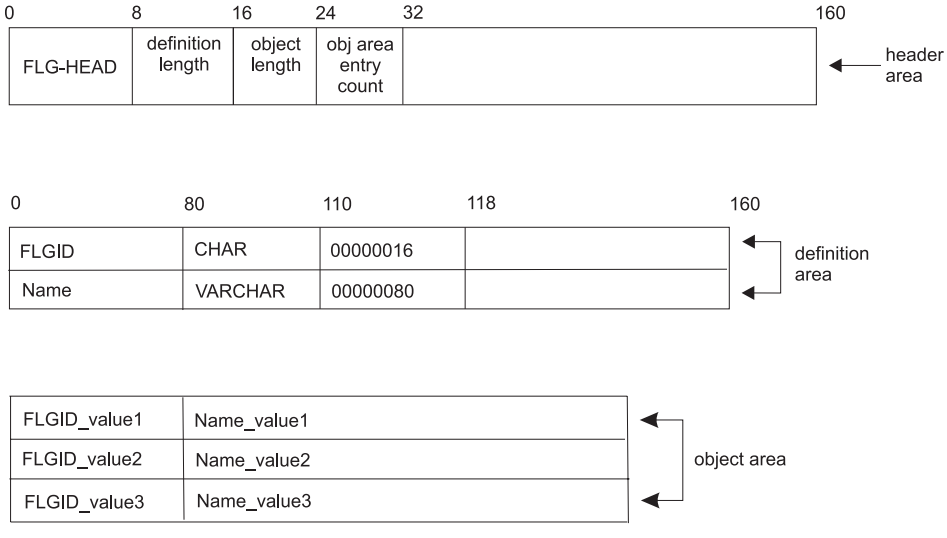

Figure 115. FLGListPrograms output structure

Usage

Freeing memory allocated for an output structure

If FLGListPrograms returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 116 on page 175 shows the C language code required to invoke the FLGListPrograms API call.

This sample code retrieves a list of programs that the object type named REPORT is associated with.

There are two programs created to use with REPORT: Read report and Update report.

```
APIRET          rc;               // reason code from FLGListPrograms
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
PFLGHEADERAREA * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE      ExtCode=0;             // extended code
  .
  .  /* set object type ID to ID of 'REPORT'                */
  .
rc = FLGListPrograms (pszObjTypeID,
                      ppListStruct,
                      &ExtCode);
```

Figure 116. Sample C language call to FLGListPrograms

Figure 117 shows the output structure for this API call.

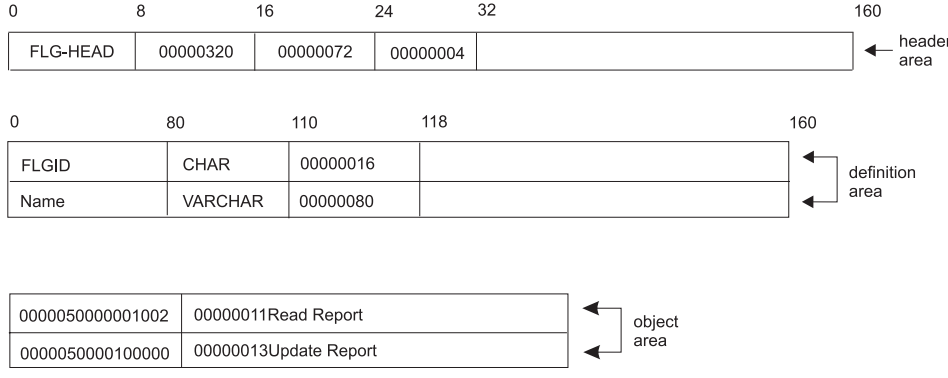

Figure 117. Sample output structure for FLGListPrograms

FLGManageCommentStatus

Sets the list of available status choices for users to assign to Comments objects they create in the information catalog using the Information Catalog Manager interface. For example, status choices might be: Open, Pending, Action required, and Closed.

Authorization

Administrator; user (FLG_ACTION_GET only)

Syntax

```
APIRET  APIENTRY   FLGManageCommentStatus( FLGOPTIONS       Action,
                                           FLGHEADERAREA  * pStatusStruct,
                                           PFLGHEADERAREA * ppStatusStruct,
                                           PFLGEXTCODE      pExtCode );
```

Parameters

Action (FLGOPTIONS) — input
Choose one of the following action options:

FLG_ACTION_GET
Retrieves a list of current status choices for Comments object instances

FLG_ACTION_UPDATE
Adds, changes, or deletes entries from the list of status choices for Comments object instances

pStatusStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the updated list of status choices for Comments object instances for FLG_ACTION_UPDATE.

ppStatusStruct (PFLGHEADERAREA) — output
Points to the output structure that contains the current list of status choices for Comments object instances for FLG_ACTION_GET.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Each time you call FLGManageCommentStatus, you must include the entire 10-entry definition area and corresponding 10 entries in the object area. Use zeros for status areas that you want to leave blank (see Figure 119 on page 178).

Freeing memory allocated for an output structure

If FLGManageCommentStatus returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGManageCommentStatus completes successfully. If FLGManageCommentStatus does not complete successfully, include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

This sample code retrieves the status structure. Figure 118 shows the C language code required to issue the FLGManageCommentStatus call.

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pStatusStruct;
FLGEXTCODE       xc=0;             // extended code

  .
  .  /*                                                */
  .
Action= Action | FLG_ACTION_GET;  //set get option
rc = FLGManageCommentStatus (Action,
                             NULL,
                             &pStatusStruct,
                             &xc);
```

Figure 118. Sample C language call to FLGManageCommentStatus

Figure 119 on page 178 shows the output structure for the FLGManageCommentStatus call in Figure 118.

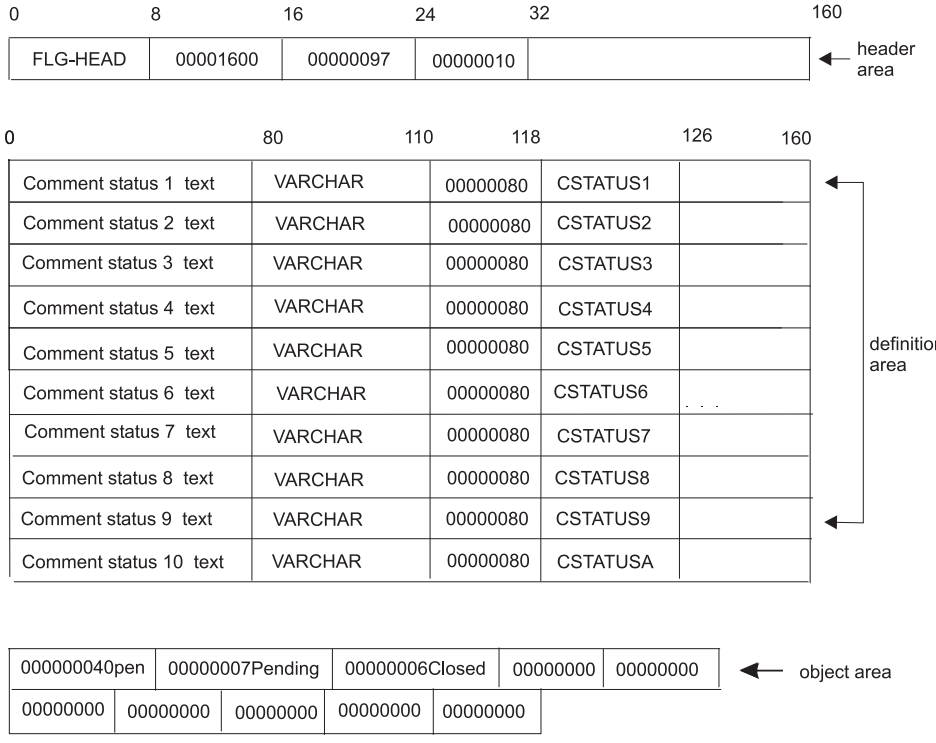

Figure 119. Sample output structure for FLGManageCommentStatus

This sample code updates the status structure with an additional status field. Figure 120 shows the C language code required to issue the FLGManageCommentStatus call.

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pStatusStruct;
FLGEXTCODE       xc=0;             // extended code

  .
  .  /*                                                */
  .
Action= Action | FLG_ACTION_UPDATE;  //update option
rc = FLGManageCommentStatus (Action,
                             pStatusStruct,
                             NULL,
                             &xc);
```

Figure 120. Sample C language call to FLGManageCommentStatus

Figure 121 shows the input structure for the FLGManageCommentStatus call in Figure 120 on page 178.

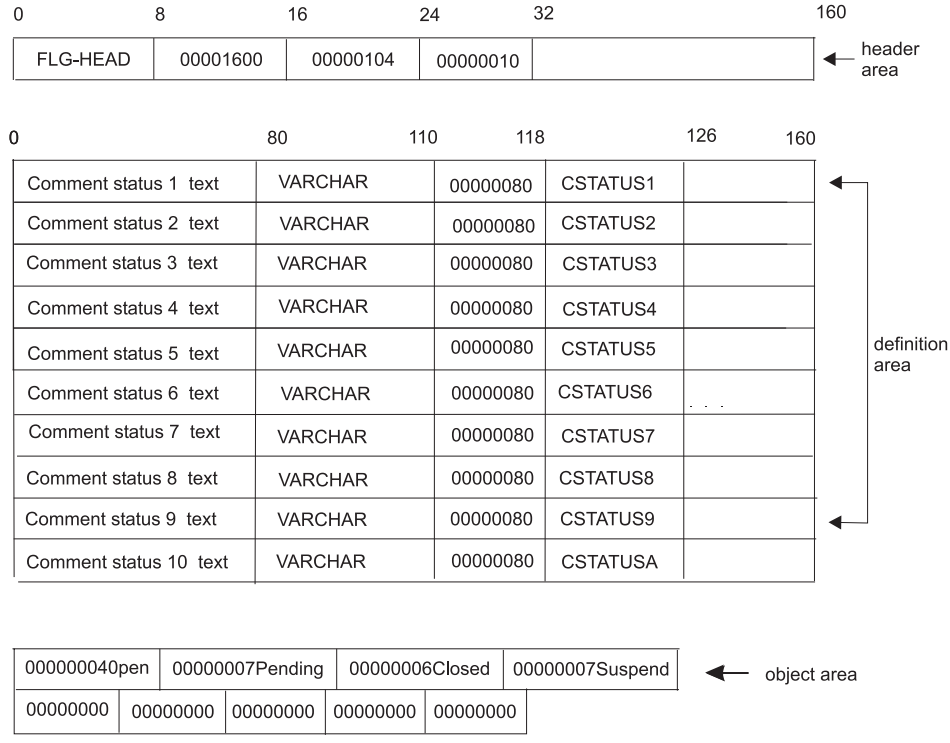

Figure 121. Sample input structure for FLGManageCommentStatus

FLGManageFlags

Queries or starts or stops recording delete history. *Delete history* is a log of delete activity that can be turned on and off.

Authorization

Administrator; user (FLG_ACTION_GET only)

Syntax

```
APIRET  APIENTRY   FLGManageFlags( FLGOPTIONS   Action,
                                   FLGOPTIONS   FlagType,
                                   UCHAR        chValue,
                                   UCHAR      * pchValue,
                                   PFLGEXTCODE  pExtCode );
```

Parameters

Action (FLGOPTIONS) — input
Choose one of the following action options:
FLG_ACTION_GET
Indicates whether logging of delete history is currently enabled or disabled
FLG_ACTION_UPDATE
Turns on or off logging of delete history

FlagType (FLGOPTIONS) — input
Indicates the flag type. This value must be FLG_HISTORY_TYPE_DELETE.

chValue (UCHAR) — input
Indicates desired flag value for FLG_ACTION_UPDATE. Choose one of the following flags:
FLG_YES
Enables logging of delete history
FLG_NO
Disables logging of delete history

pchValue (UCHAR) — output
Points to the status returned by FLG_ACTION_GET, either:
FLG_YES
Logging of delete history is enabled
FLG_NO
Logging of delete history is disabled

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)

Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGManageFlagssuccessfully updates flags. If FLGManageFlags does not update flags successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 122 shows the C language code required to issue the FLGManageFlags call. This sample code enables logging of the delete history.

```
APIRET          rc;               // reason code for API
FLGOPTIONS      Action=0;
FLGOPTIONS      Type=0;
UCHAR           chValue=FLG_YES;
FLGEXTCODE      xc=0;             // extended code

  .
  .
  .
Action= Action | FLG_ACTION_UPDATE;
Type = Type | FLG_HISTORY_TYPE_DELETE;
rc = FLGManageFlags (Action,
                     Type,
                     chValue,
                     NULL,
                     &xc);
```

Figure 122. Sample C language call to FLGManageFlags

FLGManageIcons

Creates, deletes, gets, queries, or updates representative OS/2 or Windows icons.

Authorization

Administrator; user (FLG_ACTION_GET and FLG_ACTION_QUERY only)

Syntax

```
APIRET  APIENTRY   FLGManageIcons( PSZ          pszObjTypeID,
                                   PSZ          pszIconFileID,
                                   FLGOPTIONS   InOptions,
                                   PFLGOPTIONS  pOutOptions,
                                   PFLGEXTCODE  pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Points to the 6-character, system-generated unique identifier (object type ID) of an object type for which you want to retrieve, query, create, update, or delete icons.

pszIconFileID (PSZ) — input
Contains the drive, directory path, and file name (valid for a FAT or HPFS file) of the file that contains the OS/2 or Windows icon you want to retrieve, create, or update for the specified object type. This parameter is ignored for FLG_ACTION_QUERY and FLG_ACTION_DELETE.

InOptions (FLGOPTIONS) — input
Indicates the desired action and platform options. Choose one of the following action options:

FLG_ACTION_CREATE
Adds the specified icon to the specified object type.

FLG_ACTION_DELETE
Removes the specified icon from the specified object type.

FLG_ACTION_GET
Retrieves the specified icon file.

FLG_ACTION_QUERY
Determines whether the specified icon file exists.

FLG_ACTION_UPDATE
Changes the icon for the specified object type.

Choose one of the following platform options:

FLG_PLATFORM_OS2
Manages OS/2 icons.

FLG_PLATFORM_WINDOWS
Manages Windows icons.

pOutOptions (PFLGOPTIONS) — output
Points to the status returned by FLG_ACTION_QUERY, either:
FLG_ICON_EXIST
FLG_ICON_NOTEXIST

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Prerequisite:

Before you can call FLGManageIcons, you need to call FLGCreateReg to register the object type for which you want to manage icons.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGManageIcons successfully creates, updates, or deletes icons. If FLGManageIcons does not create, update, or delete icons successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 123 on page 184 shows the C language code required to issue the FLGManageIcons call. This sample code updates a Windows icon in the Information Catalog Manager.

```
APIRET            rc;                // reason code from FLGManageIcons
UCHAR             pszObjTypeID[FLG_OBJTYPID_LEN + 1];
UCHAR             pszIconFileID[FLG_ICON_FILE_ID_MAXLEN + 1];
FLGOPTIONS        Options = 0;       // initialize option
FLGEXTCODE        xc=0;              // extended code
  .
  .  /* provide values for input parameters */
  .
Options = Options | FLG_ACTION_UPDATE | FLG_PLATFORM_WINDOWS;
rc = FLGManageIcons (pszObjTypeID,
                     pszIconFileID,
                     Options,
                     NULL,
                     &xc);
```

Figure 123. Sample C language call to FLGManageIcons

FLGManageTagBuf

Queries or resets the current delete history. *Delete history* is a log of delete activity that can be turned on and off.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGManageTagBuf( FLGOPTIONS       InOptions,
                                    PFLGOPTIONS      pOutOptions,
                                    PFLGEXTCODE      pExtCode );
```

Parameters

InOptions (FLGOPTIONS) — input
Choose one of the following options:
FLG_TAGBUF_QUERY
Queries whether the delete history log currently contains entries
FLG_TAGBUF_RESET
Removes any existing entries from the delete history log

pOutOptions (PFLGOPTIONS) — output
Points to the status returned by FLG_TAGBUF_QUERY, either:
FLG_TAGBUF_STATUS_EMPTY
FLG_TAGBUF_STATUS_NOT_EMPTY

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGManageTagBuf successfully resets the delete history. If FLGManageTagBuf does not reset the delete history successfully, include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 124 shows the C language code required to issue the FLGManageTagBuf call. This sample code deletes the current contents of the delete history.

```
APIRET           rc;               // reason code
FLGOPTIONS       Opt1=0;           //option
FLGEXTCODE       xc=0;             // extended code
  .
  .  /*                                                 */
  .
Opt1=Opt1 | FLG_TAGBUF_RESET;      //set reset option
rc = FLGManageTagBuf (Opt1,
                      NULL,        // not used.
                      &xc);
```

Figure 124. Sample C language call to FLGManageTagBuf

FLGManageUsers

Authorizes specified Information Catalog Manager users in your organization to perform the following object management tasks that are normally performed by an Information Catalog Manager administrator:

- Creating an object
- Deleting an object
- Updating an object
- Copying an object
- Exporting an object
- Associating contacts
- Updating links between objects
- Updating groupings of objects
- Associating programs with objects

FLGManageUsers also updates primary and backup administrators for the information catalog.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGManageUsers( FLGOPTIONS       Options,
                                   PFLGHEADERAREA   pListStruct,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

Action (FLGOPTIONS) — input

Choose one of the following action options:

FLG_ACTION_CREATE

Adds the specified users to the list of users authorized to perform additional object management tasks for the current information catalog.

FLG_ACTION_UPDATE

Changes the primary or backup administrator.

FLG_ACTION_DELETE

Removes the specified users from the list of users authorized to perform additional object management tasks for the current information catalog.

FLG_ACTION_LIST

Returns a list of the following:

Administrator

Backup administrator

Users authorized to perform additional object management tasks for the current information catalog.

pListStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the new, changed, or deleted user IDs.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the primary and backup administrators and all users authorized to perform additional object management tasks for the current information catalog.

Each entry in the output structure has the following information:
- USERID (8 characters)
- User Type (1 character) flag:
  - **A** USERID is the primary administrator (FLG_USERTYPE_PADMIN)
  - **B** USERID is the backup administrator (FLG_USERTYPE_BADMIN)
  - **D** USERID is a user authorized to perform additional object management tasks (FLG_USERTYPE_POWERUSER)

All users are sorted by User Type first, then USERID, in ascending order according to collating order of the underlying database management system.

When there is no output structure, the pointer to the structure is set to NULL.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Restrictions

The Information Catalog Manager only allows one primary and one backup administrator and only the administrators can invoke FLGManageUsers. If FLGManageUsers affects the logged-on administrator user ID, then the change will not take effect until the current administrator logs off.

Freeing memory allocated for an output structure

If FLGManageUsers returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGManageUsers successfully creates, updates, or deletes users. If FLGManageUsers does not create, update, or delete users successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

This sample code adds two users to the list of administrators and users who are authorized to perform additional object management tasks. Figure 125 shows the C language code required to issue the FLGManageUsers call.

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pInList;
FLGEXTCODE       xc=0;             // extended code

  .
  .  /*                                                  */
  .
Action= Action | FLG_ACTION_CREATE;
rc = FLGManageUsers (Action,
                     pInList,
                     NULL,
                     &xc);
```

Figure 125. Sample C language call to FLGManageUsers

Figure 126 on page 190 shows the input structure for the FLGManageUsers call in Figure 125.

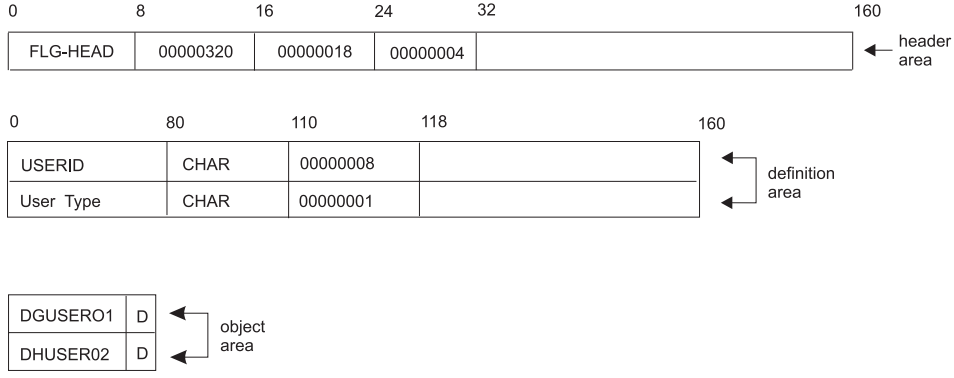

Figure 126. Sample input structure for FLGManageUsers

This sample code retrieves a current list of users who are authorized to perform additional object management tasks. Figure 127 shows the C language code required to issue the FLGManageUsers call.

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA  * ppOutList;
FLGEXTCODE       xc=0;             // extended code

  .
  .  /*                                                 */
  .
Action= Action | FLG_ACTION_LIST;
rc = FLGManageUsers (Action,
                     NULL,
                     ppOutList,
                     &xc);
```

Figure 127. Sample C language call to FLGManageUsers

Figure 128 on page 191 shows the output structure for the FLGManageUsers call in Figure 127.

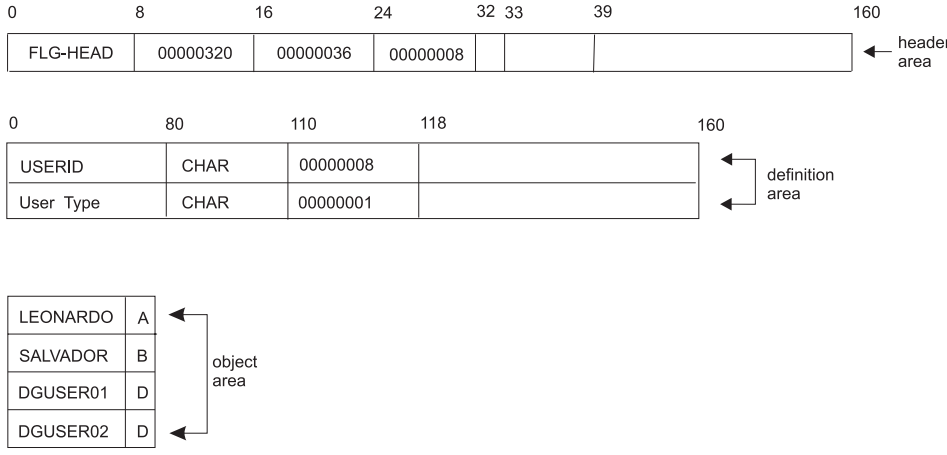

Figure 128. Sample output structure for FLGManageUsers

This sample code updates the primary administrator. Figure 129 shows the C language code required to issue the FLGManageUsers call.

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pInList;
FLGEXTCODE       xc=0;             // extended code

  .
  .  /*                                                */
  .
Action= Action | FLG_ACTION_UPDATE;
rc = FLGManageUsers (Action,
                     pInList,
                     NULL,
                     &xc);
```

Figure 129. Sample C language call to FLGManageUsers

Figure 130 on page 192 shows the input structure for the FLGManageUsers call in Figure 129. Because only the primary administrator is updated, the backup administrator remains the same.

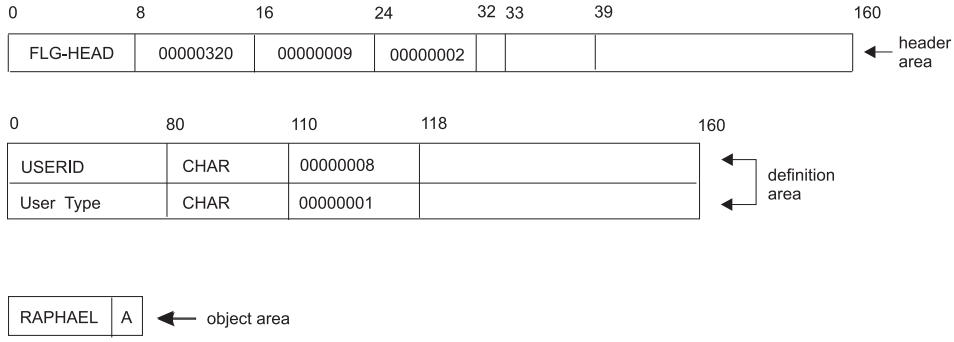

Figure 130. Sample input structure for FLGManageUsers

FLGMdisExport

Retrieves MDIS-conforming metadata from the information catalog and translates it to an MDIS-conforming file. The information catalog from which you export MDIS metadata is not limited to containing MDIS metadata, but FLGMdisExport exports only MDIS-conforming metadata.

Authorization

Administrator or authorized user

Syntax

```
APIRET  APIENTRY   FLGExport( PSZ             pszTagFileName,
                              PSZ             pszLogFileName,
                              PSZ             pszObjTypeName,
                              PSZ             pszObjectName,
                              PFLGEXTCODE     pExtCode );
```

Parameters

pszTagFileName (PSZ) — input
Name of the output tag language file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a file allocation table (FAT) or HPFS file. The target drive for this file must be a fixed disk. If you type only the file name, the Information Catalog Manager places the MDIS–conforming file on the drive and path pointed to by the DGWPATH environment variable.

The target MDIS–conforming file must not exist; the Information Catalog Manager does not overwrite existing files.

The file name and extension (excluding the drive and directories) cannot exceed 240 characters. The entire MDIS tag file name cannot exceed 259 characters.

pszLogFileName (PSZ) — input
Points to the name of the log file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a FAT or HPFS file. The target drive for the log file must be a fixed disk. The log file name cannot exceed 259 characters. If you specify only a file name, the Information Catalog Manager places the log file on the drive and path pointed to by the DGWPATH environment variable.

If the log file specified in this parameter does not exist, a new file is created. If the log file specified in this parameter already exists, then the FLGMdisExport API call appends to it.

pszObjTypeName (PSZ) — input
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.

pszObjectName (PSZ) — input
Specifies the objects you want to export. Depending on the object type you specified with the pszObjTypeName parameter, the value for pszObjectName is from three to five property values, separated by periods (.).

pszObjTypeName	pszObjectName
Database	*ServerName.DatabaseName.OwnerName*
Dimension	*ServerName.DatabaseName.OwnerName.DimensionName*
Subschema	*ServerName.DatabaseName.OwnerName.SubschemaName*
Record	*ServerName.DatabaseName.OwnerName.RecordName*
Element	*ServerName.DatabaseName.OwnerName.RecordName.ElementName*

In this list, the parts of the name are represented with their MDIS name. To find the equivalent Information Catalog Manager names, refer to Appendix B in the *Information Catalog Manager Administration Guide*.

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

For each part, enter the value of the named property for the object you want to export. You can use an asterisk (*) as a wildcard within, or instead of, any of the parts. If you enter nothing for a required part, the Information Catalog Manager 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.) If you enter nothing for an optional part, the Information Catalog Manager uses a null character when searching for objects to export.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

FLGMdisImport

Imports metadata from a file that conforms to the Metadata Interchange Specification (MDIS) into the Information Catalog Manager. The information catalog into which you import MDIS metadata must include, but is not limited to, valid MDIS object type definitions. Appendix B of the *Information Catalog Manager Administration Guide* describes the Information Catalog Manager predefined object types and how they map to MDIS.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGMdisImport( PSZ             pszTagFileID,
                                  PSZ             pszLogFileID,
                                  PFLGEXTCODE     pExtCode );
```

Parameters

pszTagFileID (PSZ) — input
Identifies the tag language file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a FAT or HPFS file. The drive cannot be a removable drive. The file name and extension, excluding the drive and directories, cannot exceed 240 characters. If you type only the file name, the Information Catalog Manager assumes that the tag language file is on the drive and path pointed to by the DGWPATH environment variable.

The file identified by pszTagFileID contains the MDIS–conforming metadata to be imported.

pszLogFileID (PSZ) — input
Specifies the location and name of the log file. This parameter is required.

This parameter contains the drive, directory path, and file name, and must be valid for a FAT or HPFS file. The drive cannot be a removable drive. If you specify only a file name, the Information Catalog Manager places the log file on the drive and path pointed to by the DGWPATH environment variable.

If the log file specified in this parameter does not exist, a new file is created. If the log file specified in this parameter already exists, then the Information Catalog Manager appends to it.

The file identified by pszLogFileID contains logging information as well as warnings and errors detected during processing of the FLGMdisImport API call.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Setting the MDIS environment

Before running MDIS import, set the MDIS environment variable:

```
SET MDIS_PROFILE=X:\SQLLIB\METADATA\PROFILES
```

where `X` is the drive where DB2 UDB is installed.

Note to those currently using MDIS with other products and Visual Warehouse 3.1: If you already had MDIS configuration and profile files, the DB2 UDB nstallation program did not overwrite them. However, before you use the MDIS function of the Information Catalog Manager for the first time, you must merge the information in the Information Catalog Manager MDIS profile and configuration files with your existing files. Complete the following steps:

1. Check the MDIS environment variable setting to locate your existing MDIS profile file (MDISTOOL.PRO) and configuration file (MDISTOOL.CFG).
2. Using a text editor, append the contents of X:\SQLLIB\METADATA\PROFILES\MDISTOOL.PRO to your existing profile file. (X is the drive where you installed DB2 UDB.)
3. Using a text editor, append the contents of X:\SQLLIB\METADATA\PROFILES\MDISTOOL.CFG to your existing configuration file. (X is the drive where you installed DB2 UDB.)

Debugging MDIS import errors

The Information Catalog Manager creates a log file when importing an MDIS-conforming file.

The log file records what happens during the import process. It includes the times and dates when the import process started and stopped. It also includes any warning or error messages for problems that occur during the process. The log file is identified by the pszLogFileID parameter.

FLGNavigate

Retrieves a list of objects contained by a specific Grouping object.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY    FLGNavigate( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

Parameters

pszFLGID (PSZ) — input
Points to the 16-character FLGID of the object instance for which contained objects will be retrieved.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure. If there is no output structure, then the pointer to the output structure is set to NULL.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGNavigate produces an output structure containing a list of objects contained by the specified object, as shown in Figure 131 on page 199.

The object area of the output structure contains a list of all the object instances contained by the specified object instance. Returned for each object instance are the values of the FLGID, the object instance external name, and the child indicator, which indicates whether the object contains any other objects.

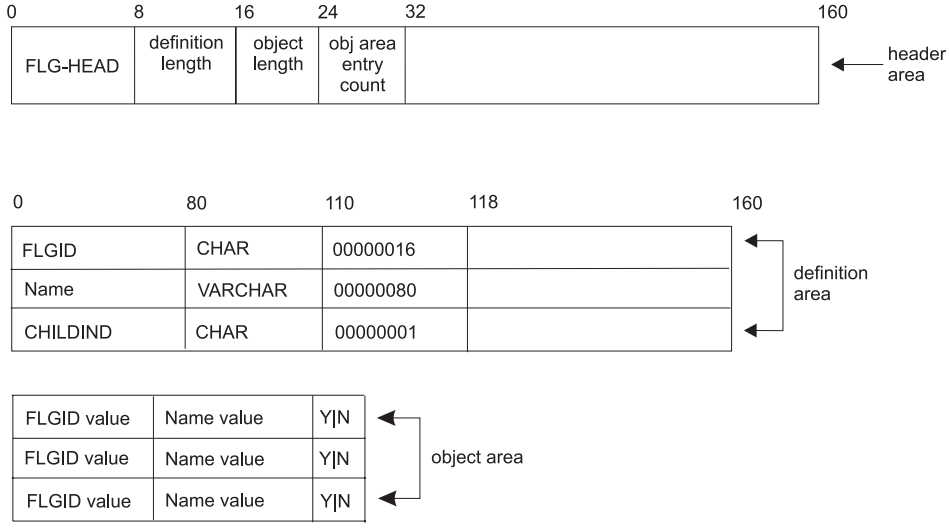

Figure 131. FLGNavigate output structure

For an explanation of the meanings of the byte offsets, see "Chapter 4. The Information Catalog Manager input and output structures" on page 31.

Usage

The output structure contains the following property values for each instance returned:

FLGID
The 16-character identifier of the object instance

Name The 80-byte external name of the object instance

CHILDIND
1-character value specifying whether an object instance contains other object instances: Y is yes, N is no

The output list is sorted by object type name, then by the 80-byte name of the object instance according to the collating order used by the underlying database management system.

The maximum number of contained object instances that can be returned by FLGNavigate is approximately 5000, depending on the storage available on your machine.

Freeing memory allocated for an output structure

If FLGNavigate returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 133 navigates through a structure of objects that contain other objects. In this example, ACCOUNTING contains three objects: ACCPAYABLE, ACCRECBLE, and GLEDGER. The structure is illustrated in Figure 132.

```
ACCOUNTING
   ACCPAYABLE
      PAYABLE1
      PAYABLE2
   ACCRECBLE
   GLEDGER
```

Figure 132. The contents of the ACCOUNTING object

Figure 133 shows the C language code required to issue the FLGNavigate API call.

```
APIRET          rc;               // reason code from FLGNavigate
UCHAR           pszFLGID[FLG_ID_LEN + 1];
PFLGHEADERAREA  * ppListStruct;
FLGEXTCODE      ExtCode = 0;           // Declare extended code
  .
  .  /* set pszParentID to FLGID of 'ACCOUNTING'          */
  .
rc = FLGNavigate (pszFLGID,
                  ppListStruct,  // pass the address of
                                 // output structure pointer
                  &ExtCode);
```

Figure 133. Sample C language call to FLGNavigate

Figure 134 on page 201 illustrates the output structure for this API call.

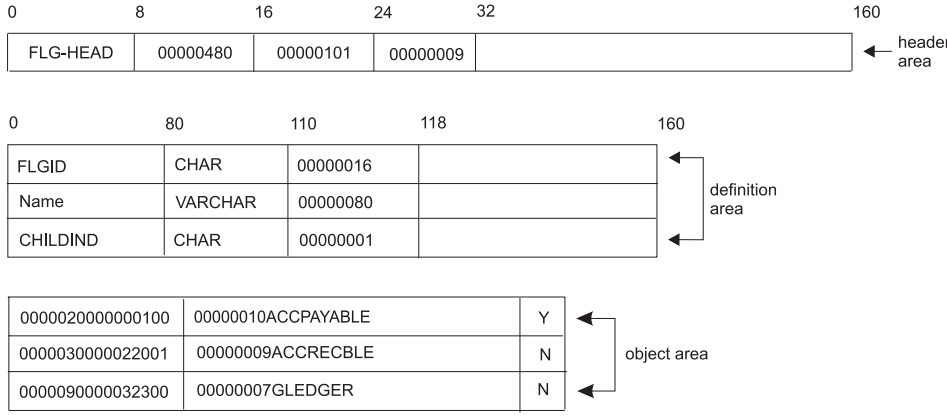

Figure 134. Sample output structure for FLGNavigate

FLGOpen

Starts an external program from the Information Catalog Manager.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGOpen( PSZ             pszPgmFLGID,
                            PSZ             pszObjFLGID,
                            PFLGEXTCODE     pExtCode );
```

Parameters

pszPgmFLGID (PSZ) — input
Points to the 16-character FLGID of the Programs object instance that contains execution information. This FLGID includes the 6-character object type ID followed by a 10-character instance ID of the Programs object.

pszObjFLGID (PSZ) — input
Points to the 16-character FLGID of a non-Program category object instance that supplies values to the parameter list. This includes the 6-character object type ID followed by a 10-character instance ID.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

To issue an FLGOpen call for a program, the program object must be set up as described in "Setting up Programs objects to start programs" on page 25.

When the program described by the Programs object starts, it uses invocation parameters provided by the identified object instance. The Information Catalog Manager removes any formatting characters entered with the invocation parameters.

Examples

Figure 135 shows the C language code required to call the FLGOpen API call. This sample code launches a program named PRINTRPT using invocation parameters supplied by an object instance named REPORT1.

```
.
 APIRET     rc;               // reason code from FLGOpen
 UCHAR      pszPgmFLGID[FLG_ID_LEN + 1];
 UCHAR      pszObjFLGID[FLG_ID_LEN + 1];
 FLGEXTCODE ExtCode = 0;           // Extended code
   .
   .  /* set pszPgmFLGID Information Catalog Manager-id of 'PRINTRPT'              */
   .  /* set pszObjFLGID to Information Catalog Manager-id of 'REPORT1'            */
   .
 rc = FLGOpen    (pszPgmFLGID,
                  pszObjFLGID,
                  &ExtCode);
```

Figure 135. Sample C language call to FLGOpen

FLGRelation

Creates or deletes the following relationships between two object instances:
- Attachment
- Contains
- Contact
- Link

Authorization

Administrator or authorized user (all relationships); user (Attachment relationships only)

Syntax

```
APIRET  APIENTRY   FLGRelation( PSZ              pszSrcFLGID,
                                PSZ              pszTrgFLGID,
                                FLGRELTYPE       RelType,
                                FLGRELOPTION     RelOpt,
                                PFLGEXTCODE      pExtCode );
```

Parameters

pszSrcFLGID (PSZ) — input
Points to the 16-character, system-generated unique identifier of the source object instance.

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

The FLGID you specify depends on the type of relationship that you want to create or delete:

Attachment relationship
FLGID of a non-Attachment category object instance to which a Comments is being attached or detached

Contact relationship
FLGID of an Elemental or Grouping category object instance for which a Contact is being defined or removed

Contains relationship
FLGID of the Grouping category container object instance

Link relationship
FLGID of an Elemental or Grouping category object instance for which a peer relationship with another object instance is to be created or deleted

pszTrgFLGID (PSZ) — input
Points to the 16-character, system-generated unique ID of the target object.

This includes the 6-character object type ID and the 10-character instance ID. The FLGID you specify depends on the type of relationship you want to create or delete:

Attachment relationship
FLGID of an Attachment category object instance being attached or detached

Contact relationship
FLGID of a Contact category object instance being defined or removed

Contains relationship
FLGID of the Elemental or Grouping category object instance being added or removed from the Grouping source container

Link relationship
FLGID of an Elemental or Grouping category object instance for which a peer relationship with another object instance is to be created or deleted

RelType (FLGRELTYPE) — input
Identifies the type of relationship being created or deleted. Valid values are:

A Attachment
C Contains
L Link
T Contact

RelOpt (FLGRELOPTION) — input
Specifies the action being performed. Valid values are:

C Create the relationship
D Delete the relationship

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Prerequisite

Before deleting an object instance, you must delete all relationships where the object instance contains other object instances.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGRelation completes successfully. If FLGRelation does not complete successfully, include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 136 shows the C language code required to call the FLGRelation API call to create a relationship defining objects contained by an object instance. In the sample code, MYBUSGP is an instance of a Business Group object type (a Grouping object), and IMAGE-A is an instance of an Image object type (an Elemental object).

```
APIRET        rc;     // Declare reason code
UCHAR  pszSrcFLGID[FLG_ID_LEN + 1];
UCHAR  pszTrgFLGID[FLG_ID_LEN + 1];
FLGRELTYPE  RelType=FLG_CONTAINER_RELATION;
FLGRELOPTION  RelOpt=FLG_CREATE_RELATION;
FLGEXTCODE    ExtCode=0;   // Declare extended code
  .
  .  /* set values for pszSrcFLGID and pszTrgFLGID      */
  .

rc = FLGRelation (pszSrcFLGID,
                  pszTrgFLGID,
                  RelType,
                  RelOpt,
                  &ExtCode);
```

Figure 136. Sample C language call to FLGRelation

FLGRollback

Deletes all information catalog changes made since the last commit point or rollback.

Authorization

Administrator and user

Syntax

```
APIRET APIENTRY FLGRollback (PFLGEXTCODE   pExtCode)
```

Parameters

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

Issue FLGRollback when your program encounters an error that might make your information catalog inconsistent.

Examples

Figure 137 shows the code that issues the FLGRollback API call.

```
APIRET        rc;           // Declare reason code from FLGRollback
FLGEXTCODE    ExtCode = 0;       // Declare extended code
  .
  .
rc = FLGRollback(&ExtCode);      // pass the address of
                             // extended code
```

Figure 137. Sample C code to invoke the FLGRollback API call

FLGSearch

Searches the information catalog to locate instances of a particular object type based on user-defined search criteria.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY    FLGSearch( PSZ              pszObjTypeID,
                               PFLGHEADERAREA   pSelCriteriaStruct,
                               PFLGHEADERAREA * ppListStruct,
                               PFLGEXTCODE      pExtCode );
```

Parameters

pszObjTypeID (PSZ) — input
Indicates any 6-character Information Catalog Manager object type ID that you want to search for.

pSelCriteriaStruct (PFLGHEADERAREA) — input
Points to an input structure that contains the property specifications and values of the search criteria.

If this value is NULL, then the Information Catalog Manager returns all instances of the specified object type.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure containing a list of selected object instances resulting from the search.

Each instance has the following information:
- FLGID (16 characters)
- Name (80 characters)

All instances are sorted by the 80-byte external name (value of Name) in ascending order according to the collating order of the underlying database management system.

The maximum number of object instances that can be returned by FLGSearch is approximately 5000, depending on the storage available on your machine.

If there is no output structure, then the pointer to the output structure is set to NULL.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See

"Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGSearch, you must define the input structure shown in Figure 138. This structure contains only the header area and the definition area.

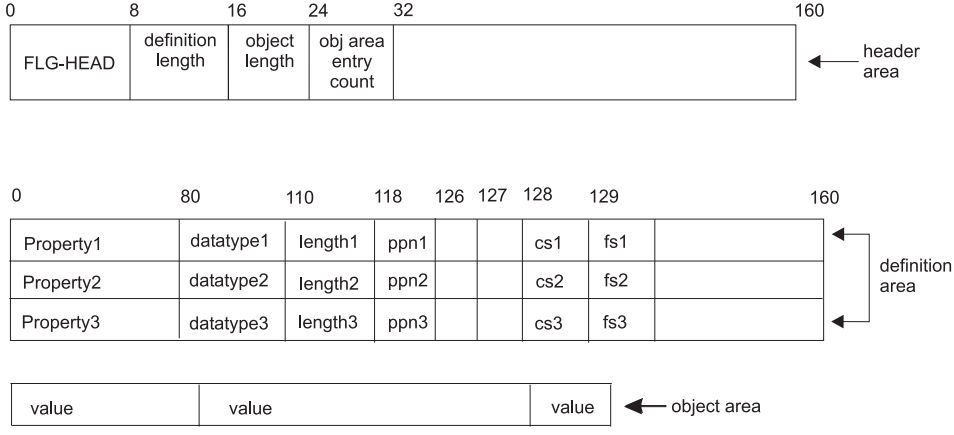

Figure 138. FLGSearch input structure

The definition area for the FLGSearch input structure must be specified as shown in Figure 138, although you can specify any and all of the properties defined for the object type. You must provide a corresponding search criteria value in the object area for each property specified in the definition area. For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API output structure" on page 51.

When the database is DB2 UDB for OS/390, the maximum length for search criteria is 254.

Output structure

FLGSearch produces an output structure containing a list of objects retrieved using the search criteria, as shown in Figure 139 on page 210.

The object area of the output structure contains a list of all the object instances that match the input search criteria. The returned object instances are identified by the values of the FLGID and object instance external name.

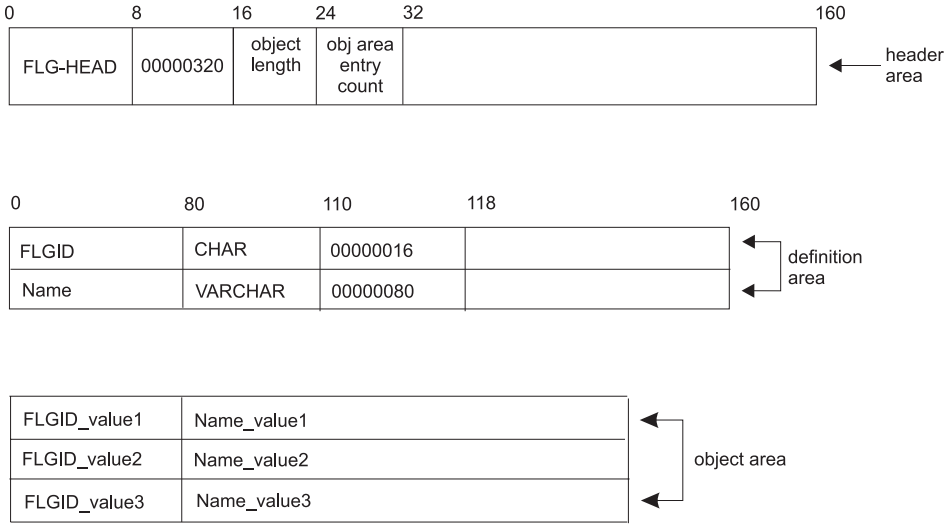

Figure 139. FLGSearch output structure

Usage

FLGSearch searches for instances of only one object type. To search for instances of all object types, use the FLGSearchAll API call.

To search for instances of more than one object type, but not all object types, call FLGSearch for each object type that you want to search.

The input structure contains the property specifications and values of the search criterion:

- Any of the object's properties can be specified as a search criterion property.
- When more than one property is specified, the properties are linked with an AND operator to produce the search criteria.
- Any blanks you include, except trailing blanks on nonCHAR data types, are considered as part of the search criterion
- You can include *wildcard* characters in the search criterion. These characters allow you to specify a pattern you are trying to locate in the values for a given property. The database supports two wildcard characters:

  % Represents zero or more characters

  _ Represents one character

Although you can use different wildcard characters in the user interface, you can only use the % and _ characters with FLGSearch.

Because DB2 databases treat trailing blanks as significant, you should include a wildcard at the end of search criteria on CHAR type properties, otherwise you might receive less objects than you expected from the call to FLGSearch.

If you include wildcard characters in the search criterion, you must set the fuzzy-search flag (fs) to Y.

– You must specify values for the following flags in the definition area:

cs Case-sensitivity flag in byte 128. Valid values are Y for case sensitive, N for not case sensitive.

If your information catalog is located on DB2 UDB for OS/390 and:
- Was created with all uppercase values (the default), then the case-sensitivity flag must be N.
- Was created with mixed-case values, then the case-sensitivity flag must be Y.

fs Fuzzy search flag in byte 129. Valid values are Y for fuzzy search, N for not a fuzzy search. This value must be Y if wildcards (% or _) are included in the search criterion.

Controlling updates to your information catalog

FLGSearch commits changes to the database. Your program should issue FLGCommit or FLGRollback before issuing FLGSearch to ensure that the Information Catalog Manager does not also commit unexpected changes that occurred before the FLGSearch call.

Freeing memory allocated for an output structure

If FLGSearch returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

FLGSearch: Example 1

The sample code in Figure 140 on page 212 performs a search for glossary instances. This search is an *exact search* because the fuzzy search flag in byte 129 of the definition area is set to N, as shown in Figure 141 on page 212. However, the case of the characters in the values (uppercase or lowercase) is not significant because the case-sensitivity flag in byte 128 is set to N. You must have already found the object type identifier using an FLGListObjTypes or FLGGetType call.

```
APIRET          rc;             // reason code from FLGSearch
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
PFLGHEADERAREA  pSelCriteria;      // search criterion input structure pointer
PFLGHEADERAREA  * ppListStruct; // pointer to search result pointer
FLGEXTCODE      ExtCode = 0;         // Declare extended code
  .
  .  /* provide values for input parameters */
  .
strcpy (pszObjTypeID, "000006");

rc = FLGSearch (pszObjTypeID,     // Information Catalog Manager object type ID
                pSelCriteria,        // input structure pointer
                ppListStruct, // pass the address of
                              // output structure pointer
                &ExtCode);
```

Figure 140. Sample C language call to FLGSearch

Figure 141 shows the search condition input structure(pointed to by pSelCriteria) that carries the property and value information for the search.

The case sensitivity flag at byte 128 and the fuzzy search flag at byte 129 of the definition area must be set to N, because the user wants an exact search, but is not concerned about the case of the property value.

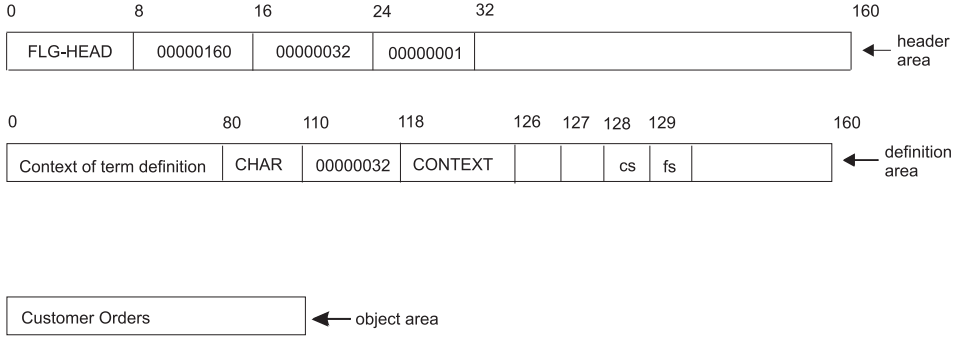

Figure 141. Sample input structure for FLGSearch

Figure 142 on page 213 shows the output structure (ppListStruct points to the address of the pointer to this output structure) that carries glossary instances as the search result.

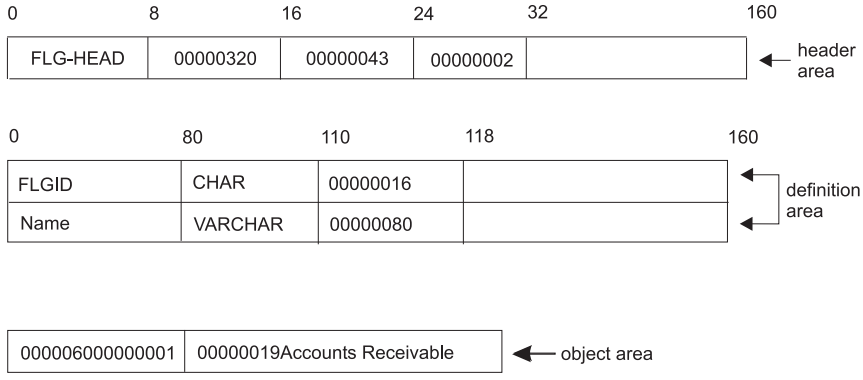

Figure 142. Sample output structure for FLGSearch

The CONTEXT value `Customer Orders` is used as the search criterion. Any glossary instance with this CONTEXT value is returned in the output structure. Because the case-sensitivity flag is set to `N`, even CONTEXT values like `customer orders` or `CUSTOMER ORDERS` would have been returned if they existed.

FLGSearch: Example 2

This example shows how your program can use fuzzy searches to locate instances that contain values fitting a pattern.

The values specified in the input structure shown in Figure 143 specify a wildcard search for glossary instances that contain the letters `metadata`. The multiple-character wildcards (%) indicate where any other characters can occur in the value and still fit the search criterion.

Figure 143 shows the input structure(pointed to by pSelCriteria) that your program passes to FLGSearch.

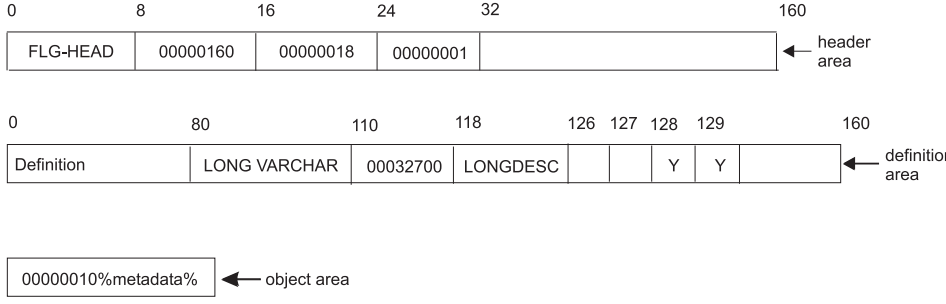

Figure 143. Sample input structure for FLGSearch

Because this is a wildcard search, the fuzzy search flag at byte 129 must be set to `Y`. If the fuzzy search flag is set to N, then the % character becomes a literal part of the search criterion; that is, any instances that are returned must have % in the specified property value.

The case sensitivity flag at byte 128 of the definition area is set to Y because the case of metadata is significant in this example. Figure 144 shows the output structure (ppListStruct points to the address of the pointer to the output structure) that carries glossary instances as the search result.

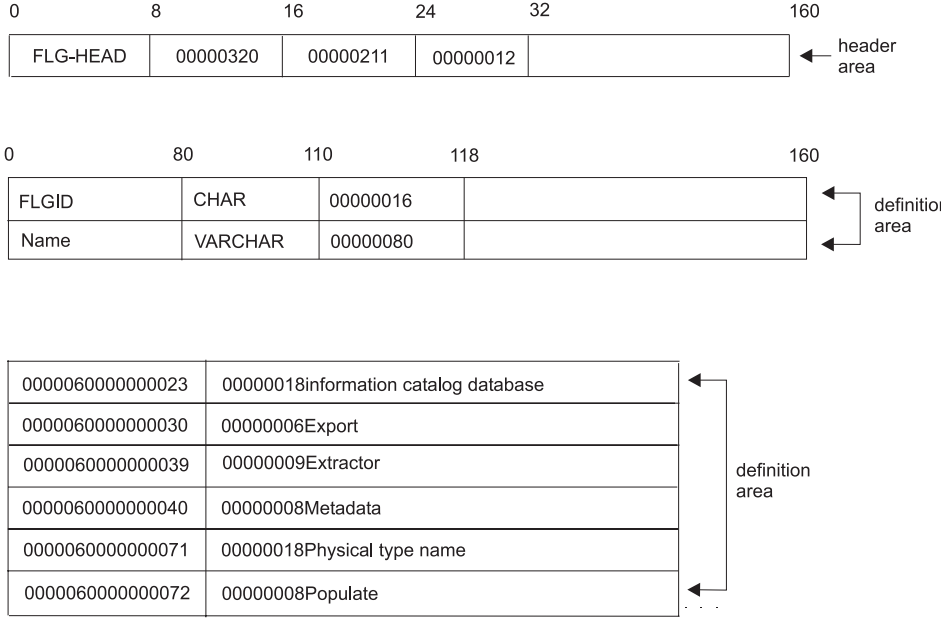

Figure 144. Sample output structure for FLGSearch

The value of the Definition property, `%metadata%`, is used as the search criterion. Any glossary instance with a Definition property value containing `metadata` is returned in the output structure. Because the case sensitivity flag is set to Y, all instances found in the example also match the case of `metadata`.

FLGSearch: Example 3

This example shows how your program can use fuzzy searches to locate instances that contain values fitting a pattern.

The values specified in the input structure shown in Figure 145 on page 215 uses the single-character wildcard (_) to search for glossary instances that have the specified property value with only one variable character.

Figure 145 shows the input structure(pointed to by pSelCriteria) that your program passes to FLGSearch.

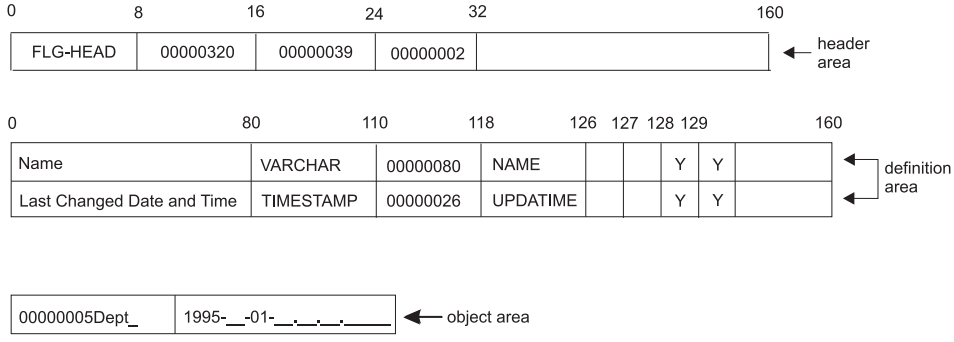

Figure 145. Sample input structure for FLGSearch

Because the search criterion contains the single-character wildcard (_), the fuzzy search flag at byte 129 must be set to Y. If the fuzzy search flag is set to N, the Information Catalog Manager assumes that _ is a literal part of the search criterion, and only returns object instances that have _ as part of the specified property value.

In this example, the values for both NAME and UPDATIME are used as the search criterion.

- The specified NAME value `Dept_` means search for instances starting with Dept and ending with an unknown character. This value contains five characters.
- Values for year and day are provided for the time stamp data type property UPDATIME. The UPDATIME values with the year `1995` and the day `01` are linked using the AND operator with the value of NAME to construct the search criteria which determine whether an object instance is returned. Both the UPDATIME value and the NAME value must match the search criterion before the Information Catalog Manager returns the object instance.

Figure 146 on page 216 shows the output structure (ppListStruct points to the address of the pointer to the output structure) that carries glossary instances as the search result.

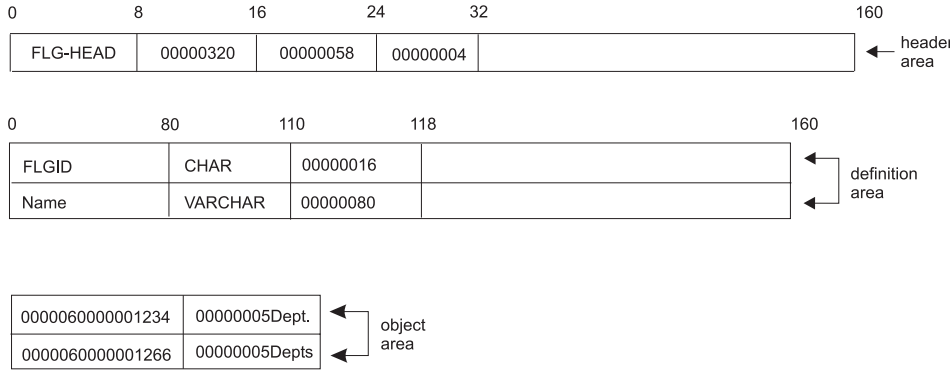

Figure 146. Sample output structure for FLGSearch

Any glossary instance with `Dept` as the prefix for the five-character Name value and updated on the first day of each month in year 1995 is returned in the output structure.

FLGSearchAll

Searches all object types in the information catalog to locate any instances that have instance names (value of Name property) that match the search criterion.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGSearchAll( PFLGHEADERAREA   pSelCriteriaStruct,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

Parameters

pSelCriteriaStruct (PFLGHEADERAREA) — input
Points to an input structure.

The structure contains the property specification and value of the search criterion. Only the value of the object instance's external name (Name) can be used as the search criterion with FLGSearchAll.

If pSelCriteriaStruct is set to NULL, then the Information Catalog Manager returns all instances in the information catalog up to a maximum of approximately 5000.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure containing a list of selected object instances resulting from the search. If there is no output structure, then the pointer to the output structure is set to NULL. Each instance has the following information:

- FLGID (16 characters)
- Name (80 characters)

All instances are first sorted by object type name, then by the instance external name (value of Name) in ascending order according to the collating order of the underlying database management system.

The maximum number of object instances that can be returned by FLGSearchAll is approximately 5000, depending on the storage available on your machine.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To specify search criterion for FLGSearchAll, you must define the following input structure. This structure contains the header area, the definition area, which can contain only the Name property, and the object area.

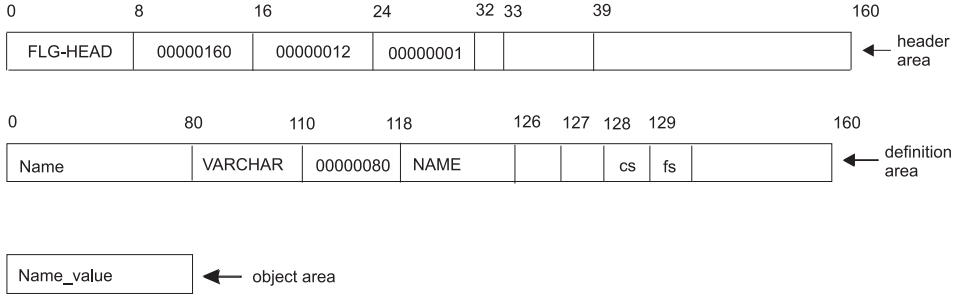

Figure 147. FLGSearchAll input structure

The definition area for the FLGSearchAll input structure must be specified exactly as shown in Figure 147. For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API input structure" on page 32.

Output structure

FLGSearchAll produces an output structure containing a list of objects retrieved using the search criterion, as shown in Figure 148 on page 219.

The object area of the output structure contains a list of all the object instances that match the input search criteria. The returned object instances are identified by the values of the FLGID and object instance external name.

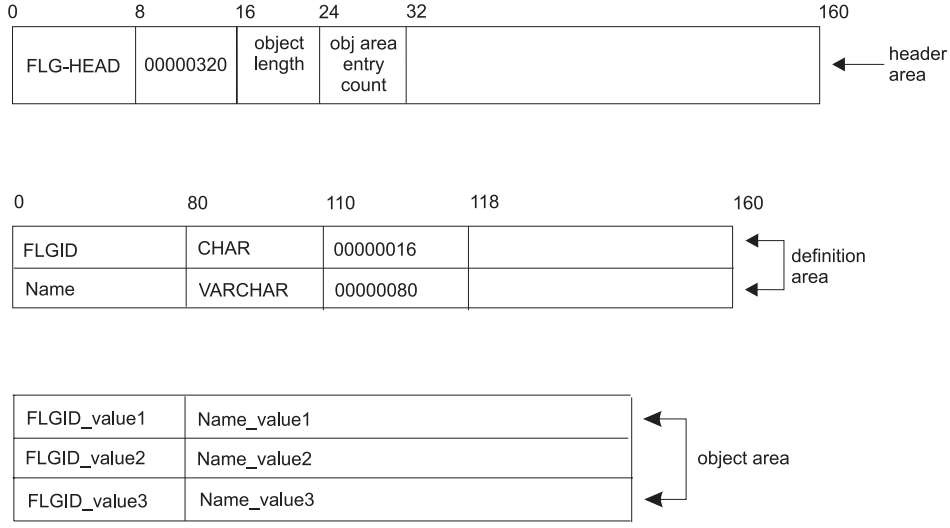

Figure 148. FLGSearchAll output structure

Usage

Only the value of the object instance's external name (`Name`) can be used as the search criterion. No other property values can be used with FLGSearchAll. If you need to use the values of other properties in your search criterion, use FLGSearch (see "FLGSearch" on page 208).

You can include *wildcard* characters in the search criterion. These characters allow you to specify a pattern you are trying to locate in the values for a given property. The database supports two wildcard characters:

% Represents zero or more characters

_ Represents one character

Although you can use different wildcard characters in the user interface, you can only use the `%` and _ characters with FLGSearchAll.

If you include wildcard characters in the search criterion, you must set the fuzzy-search flag (fs) to Y.

You must specify values for the following flags in the definition area:

cs Case-sensitivity flag in byte 128. Valid values are Y for case sensitive, N for not case sensitive.

If your information catalog is located on DB2 UDB for OS/390 and:

– Was created with all uppercase values (the default), then the case-sensitivity flag must be N.

– Was created with mixed-case values, then the case-sensitivity flag must be Y.

fs Fuzzy search flag in byte 129. Valid values are Y for fuzzy search, N for not a fuzzy search. This value must be Y if you want to search using wildcards (% or _) are included in the search criterion.

Controlling updates to your information catalog

FLGSearchAll commits changes to the database. Your program should issue FLGCommit or FLGRollback before issuing FLGSearchAll to ensure that the Information Catalog Manager does not also commit unexpected changes that occurred before the FLGSearchAll call.

Freeing memory allocated for an output structure

If FLGSearchAll returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 149 shows the C language code required to invoke the FLGSearchAll. This sample code searches for a name across all object type instances.

```
APIRET           rc;                // reason code from FLGSearchAll
PFLGHEADERAREA   pSelCriteria;         // search criterion input structure pointer
PFLGHEADERAREA  * ppListStruct;  // pointer to search result pointer
FLGEXTCODE       ExtCode = 0;            // Declare extended code
  .
  .  /* provide values for input parameters */
  .

rc = FLGSearchAll ( pSelCriteria,        // input structure pointer
                ppListStruct, // pass the address of
                                // output structure pointer
                &ExtCode);
```

Figure 149. Sample C language call to FLGSearchAll

Figure 150 on page 221 shows the search condition input structure (pointed to by pSelCriteria) that carries the property and value information for the search.

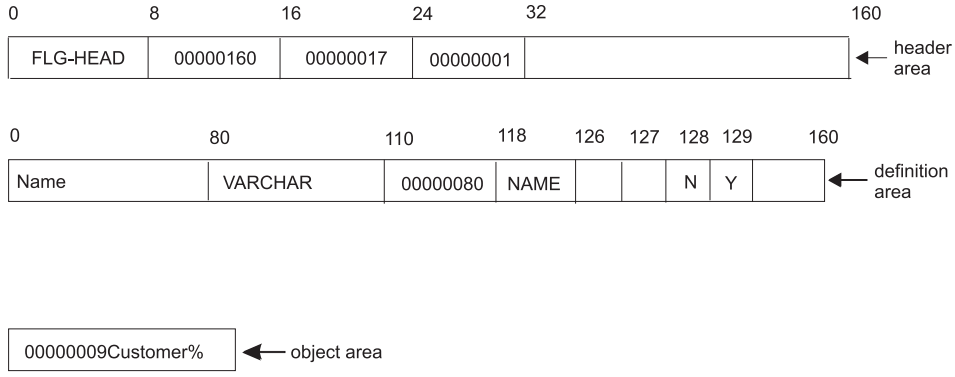

Figure 150. Sample input structure for FLGSearchAll

In this example, you want to perform a fuzzy search using wildcard characters in the search criterion, so the fuzzy search flag at byte 129 of the definition area is set to `Y`.

The case-sensitivity flag at byte 128 of the definition area is set to N, because the user does not need case sensitivity in the search criterion. Figure 151 on page 222 shows the output structure (ppListStruct points to the address of the pointer to this output structure) that carries the Information Catalog Manager objects as the search result.

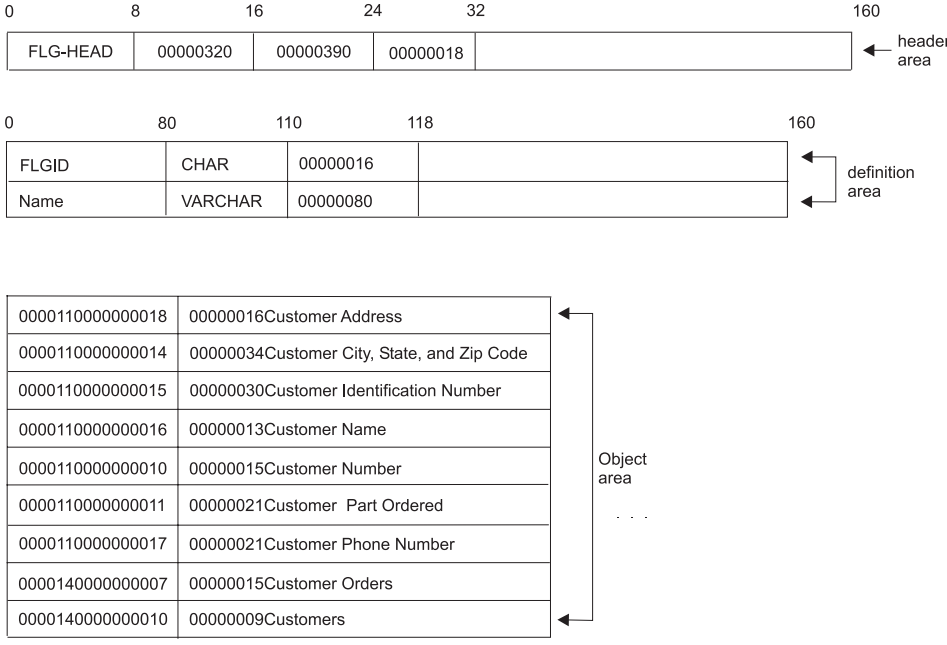

Figure 151. Sample output structure for FLGSearchAll

The specified partial object instance name is used by the nine instances in two different object types.

FLGTerm

Ends the Information Catalog Manager API DLL environment, disconnects from the database manager, and frees all associated system resources.

Authorization

Administrator or user

Syntax

```
APIRET APIENTRY FLGTerm (PFLGEXTCODE   pExtCode )
```

Parameters

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

When your program calls FLGTerm, the Information Catalog Manager automatically commits any uncommitted changes to the information catalog database. If any changes need to be rolled back, your program should call FLGRollback before calling FLGTerm to exit the Information Catalog Manager.

If the Information Catalog Manager encountered a severe error while trying to roll back the database, FLGTerm will encounter an error while shutting down the Information Catalog Manager and trying to release resources. If the person using your program is logged on as an administrator when the FLGTerm call fails, that person might need to use the Information Catalog Manager CLEARKA utility to log off the administrator user ID.

Examples

Figure 152 on page 224 shows the C language code required to invoke the FLGTerm API call. This sample code stops the Information Catalog Manager API DLL.

```
.

APIRET         rc;               // Reason code
FLGEXTCODE     ExtCode = 0;           // Extended code

.
.  // FLGInit()
.  // calls to the FLG API

rc = FLGTerm ( &ExtCode );
```

Figure 152. Sample C language call to FLGTerm

FLGTrace

Sets the level of information about the Information Catalog Manager function written in the trace (.TRC) file.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGTrace( FLGTRACEOPTION   TraceOpt,
                             PFLGEXTCODE      pExtCode );
```

Parameters

TraceOpt (FLGTRACEOPTION) — input
Indicates the desired trace option. Valid options are:

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

1 Include entry and exit records of the highest-level Information Catalog Manager functions.

2 Include extremely granular entry and exit records of the Information Catalog Manager functions.

3 Include input and output parameters (excluding input or output structure)

4 Include all input or output structures that are passed to and used by the Information Catalog Manager, including SQLCA information passed to and used by the underlying database management system.

Constants for these values are defined in the Information Catalog Manager API header file, DG2API.H.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

The name of the trace file is the name of the information catalog you are using with the extension of `.TRC`.

When you use trace files to debug your programs, levels 0 and 4 are most likely to be useful to you.

Level 0

Returns information explaining the functions that the Information Catalog Manager is performing.

When the Information Catalog Manager encounters an error, it inserts the reason code and extended code for that error into the trace file as the New Reason Code and the New Extended Code. The trace file also contains an Old Reason Code and an Old Extended Code, which contain the reason code that was returned before the error occurred. Any messages that the Information Catalog Manager produces in the trace file.

Level 4

Returns the same information as for Level 0, more detailed functional information about the Information Catalog Manager, and information about the data structures passed to and from the Information Catalog Manager, including input structures, output structures and SQLCA structures from the database.

Tracing the contents of these structures can be valuable when you need to determine the cause of data errors or ensure that the contents of an input or output structure is being produced or read properly.

For more information about using trace files, see the *Information Catalog Manager Administration Guide*.

Examples

Figure 153 on page 227 shows the C language code required to invoke the FLGTrace API call. This sample code sets the level of tracing from an information application.

```
.
 FLGTRACEOPTION TraceOpt   = FLG_TRACELEVEL_1; // Turn on Entry/Exit Tracing
 FLGTRACEOPTION TraceReset = FLG_TRACELEVEL_0; // Reset to default level
 APIRET         rc;                          // reason code
 FLGEXTCODE     ExtCode = 0;                      // Extended code

 .
 .  // FLGInit()
 .  // calls to the FLG API

 rc = FLGTrace ( TraceOpt,
                 &ExtCode );
 .
 .  // Check rc and ExtCode
 .
 .  // More API calls
 .

 rc = FLGTrace ( TraceReset,
                 &ExtCode );
```

Figure 153. Sample C language call to FLGTrace

FLGUpdateInst

Alters one or more property values for a specific object instance.

Authorization

Administrator or authorized user (all object types); user (Comments object type only)

Syntax

```
APIRET  APIENTRY    FLGUpdateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PFLGEXTCODE      pExtCode );
```

Parameters

pObjInstStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the property specifications and values of the database object being updated.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGUpdateInst, you must define the input structure shown in Figure 154 on page 229. This structure contains the header area, the definition area, and the object area.

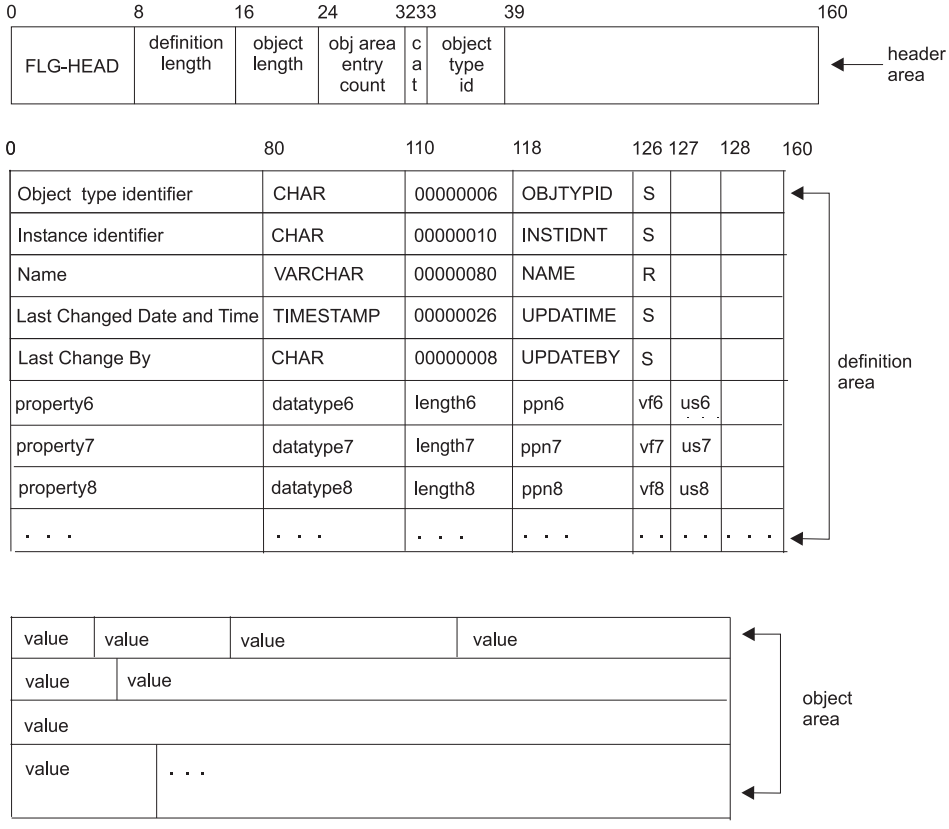

Figure 154. FLGUpdateInst input structure

For an explanation of the meanings of the byte offsets, see "The Information Catalog Manager API input structure" on page 32.

Usage

Prerequisites

Before issuing an FLGUpdateInst call, you must issue either an FLGCreateInst call or an FLGGetInst call to obtain the property specifications and values of the instance being modified.

Input requirements

Header area:

- All of the information shown in the header record is required.
- The value for the object type identifier must be the same in the header record (bytes 33 through 38) as in the object area (first item in the object area).

Definition area:

The definition area can contain any or all of the defined properties of the object type for which you are updating an object instance. The following rules apply:

- You must first specify all five of the Information Catalog Manager required properties in the following order: OBJTYPID, INSTIDNT, NAME, UPDATIME, and UPDATEBY.
- You must specify all UUI properties.
- The Information Catalog Manager compares the values for all specified properties to the object type definition for the following specifications:

  Data type

  Data length

  Property short name

  Value flag

  UUI number

Object area:

- The object type in the HANDLES property (when specified) must exist in the information catalog and be a non-Program object type. Any properties specified in the PARMLIST property must be a property of the object type specified in HANDLES. For more information, see "Setting up Programs objects to start programs" on page 25.
- If a value is not specified for a required property (defined with an R in column 126 of the definition area) the appropriate space in the object area must be initialized as follows:

Data type	Initialized to
CHAR	Not-applicable symbol followed by blanks for the length of the property
TIMESTAMP	Set to the largest allowable value: `9999-12-31-24.00.00.000000`
VARCHAR LONG VARCHAR	00000001; the length field, specified in 8 bytes, followed by the not-applicable symbol

- Values for the OBJTYPID and INSTIDNT properties identify the instance being updated, and therefore must be present.
- Values for the UPDATIME and UPDATEBY properties are system generated and therefore should not be modified by the user. If you issue an FLGGetInst call before issuing this FLGUpdateInst call, the object area can contain values for these two system-generated properties. This does not cause an error, but when the instance is updated, the system replaces the values of these two properties.

Trailing blanks are automatically removed from object area values that have VARCHAR or LONG VARCHAR data types and the length field is adjusted accordingly.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGUpdateInst completes successfully. If FLGUpdateInst does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 155 shows the C language code required to invoke the FLGUpdateInst API call.

This sample code updates the object instance named Quality Group that was defined in the FLGCreateInst example. The update modifies the value for the short description property, Sdesc.

```
APIRET         rc;                    // Declare reason code
PFLGHEADERAREA  pObjInstStruct;       // Pointer to the input structure
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  .  /* updating pObjInstStruct object Instance by   */
     /*                            providing an updated input structure */
  .

rc = FLGUpdateInst (pObjInstStruct,   // Pointer to updated input structure
                    &ExtCode);             // Pass pointer to extended code
```

Figure 155. Sample C language call to FLGUpdateInst

Figure 156 on page 232 shows the input structure (pointed to by the ″pObjInstStruct″ pointer in the C code) that carries the property and value information for the object instance to be updated.

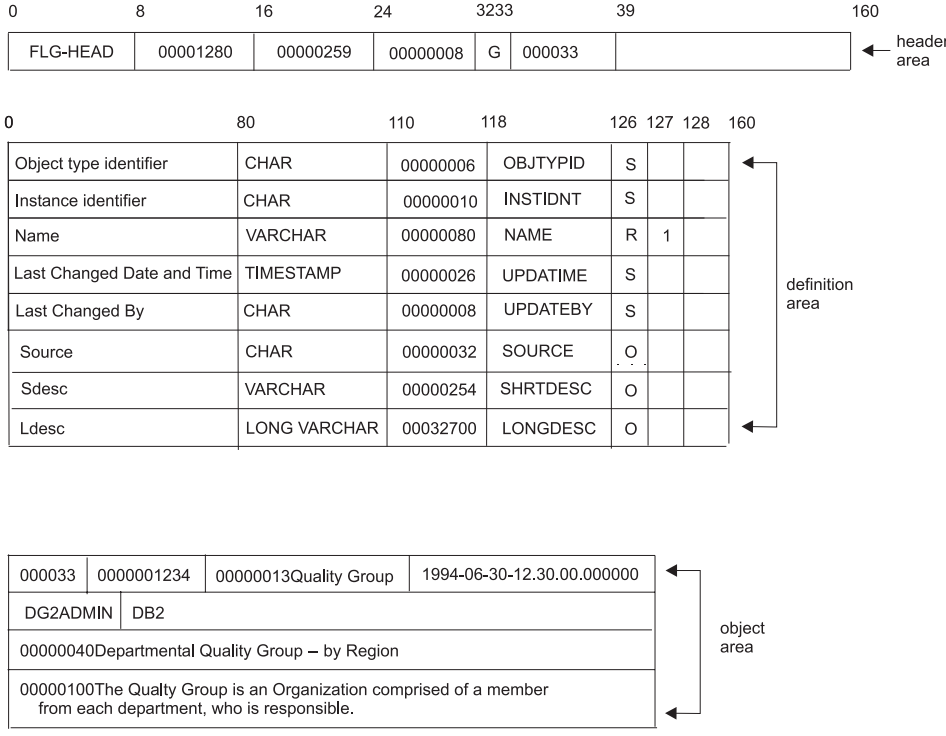

Figure 156. Sample input structure for FLGUpdateInst

The values in the object area that are not system-generated (the value at byte 126 is not S) can be modified:

- NAME
- SOURCE
- SHRTDESC
- LONGDESC

When you use FLGUpdateInst, you can omit properties and values that you are not modifying.

In this example, the Sdesc value is updated. Modifying the Sdesc value affects its length also. Therefore, the 8-character length field that precedes the Sdesc field in the object area is modified from 27 to 40. The object Length value in the header record is changed from 246 to 259.

When FLGUpdateInst completes, the value for UPDATEBY is modified to contain the user ID used to update the instance, and UPDATIME is modified to contain the time stamp of the update.

FLGUpdateReg

Modifies registration information in the information catalog for a specific object type.

This action does *not* update the object type itself; it updates the *registration* information for the object type.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGUpdateReg( PFLGHEADERAREA   pObjRegStruct,
                                 PSZ              pszIconFileID,
                                 PFLGEXTCODE      pExtCode );
```

Parameters

pObjRegStruct (PFLGHEADERAREA) — input
Points to the input structure that contains the property specifications and values of the object type registration being updated.

pszIconFileID (PSZ) — input
Contains the drive, directory path, and file name of the file that contains the OS/2 ICON for the object type registration being updated. If this parameter is NULL, then no change is made to the ICON. If specified, the OS/2 ICON is added to the object type registration if an ICON does not currently exist or replaces any existing ICON for the registration.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Input structure

To use FLGUpdateReg, you must define the input structure shown in Figure 157 on page 234. This structure contains only the header area and the definition area.

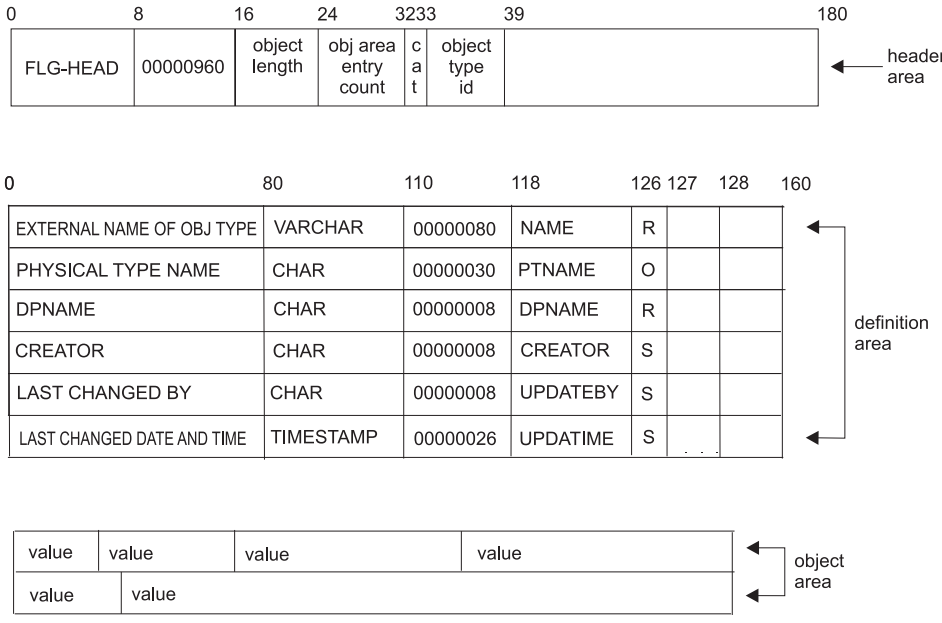

Figure 157. FLGUpdateReg input structure

Usage

Restrictions

- The registration information stored in the information catalog by FLGCreateReg consists of registration values, such as DP NAME, Physical Type Name, External Name, and Icon, which describe the object type. FLGUpdateReg can only update the External Name and Icon information.
- You can only update an OS/2 icon using FLGUpdateReg. To update a Windows icon, use FLGManageIcons (see "FLGManageIcons" on page 182).
- After you define the object type using FLGCreateReg, you can issue FLGUpdateReg or FLGManageIcons calls to change the icon that is associated with the object type, or add an icon association if one was not defined originally. You can also use FLGManageIcons to remove an icon from an object type.

Prerequisites

Before issuing an FLGUpdateReg call, you must obtain the current values of the registration information. You can either save this information from the original FLGCreateReg call, or issue an FLGGetReg call for the object type registration being modified.

Input requirements

Header area

All of the information shown in the header record is required.

Definition area

- The definition area must contain definitions for each of the six registration properties. The definitions for each of these registration properties are fixed, and all specifications other than those for the property name must be exactly as shown in Figure 157 on page 234. The property name is also fixed, but might be translated from the English property name illustrated in the example into any one of the supported languages.
- The properties (as identified by their property short names) must be specified in the following order in the definition and object area:
  1. NAME
  2. PTNAME
  3. DPNAME
  4. CREATOR
  5. UPDATEBY
  6. UPDATIME

  These properties are explained in "FLGCreateReg" on page 84.

Object area

Only the value for NAME (EXTERNAL NAME OF OBJ TYPE) can be updated. The NAME value must be unique within the local information catalog.

The remaining property values cannot be modified. CREATOR, UPDATEBY, and UPDATIME are system-generated values. DPNAME and PTNAME are the unique identifiers of the object type, and cannot be updated. Values for system-generated properties are generated when the object type itself is created or appended.

The value for DPNAME must be specified and match the DPNAME of the current object registration associated with the object type ID in the header area.

Controlling updates to your information catalog

To keep your program as synchronized as possible with your information catalog, you should include a call to FLGCommit (see "FLGCommit" on page 74) after FLGUpdateReg completes successfully. If FLGUpdateReg does not complete successfully, you should include a call to FLGRollback (see "FLGRollback" on page 207).

Examples

Figure 158 on page 236 shows the C language code required to invoke the FLGUpdateReg API call. This sample code updates the object type registration

for the MYIMAGE object type. The update modifies the value for the external name property, NAME.

```
APIRET         rc;                    // Declare reason code
PFLGHEADERAREA pObjRegStruct;         // Pointer to the input structure
UCHAR          pszIconFileID[FLG_ICON_FILE_ID_MAXLEN+1]; // Path/File name of ICON
FLGEXTCODE     ExtCode=0;                  // Declare extended code
  .
  .  /* updating pObjRegStruct object type     */
     /*                  registration by providing an updated input structure */
  .
strcpy (pszIconFileID,"Y:\\FLGICON2.ICO");

rc = FLGUpdateReg (pObjRegStruct,    // Pointer to updated Input Structure
                   pszIconFileID,    // Path/File name of file containing the ICON
                   &ExtCode);             // Pass pointer to extended code
```

Figure 158. Sample C language call to FLGUpdateReg

Figure 159 shows the input structure (pointed to by the pObjRegStruct pointer in the C code) that carries the property and value information for the object type registration information to be updated.

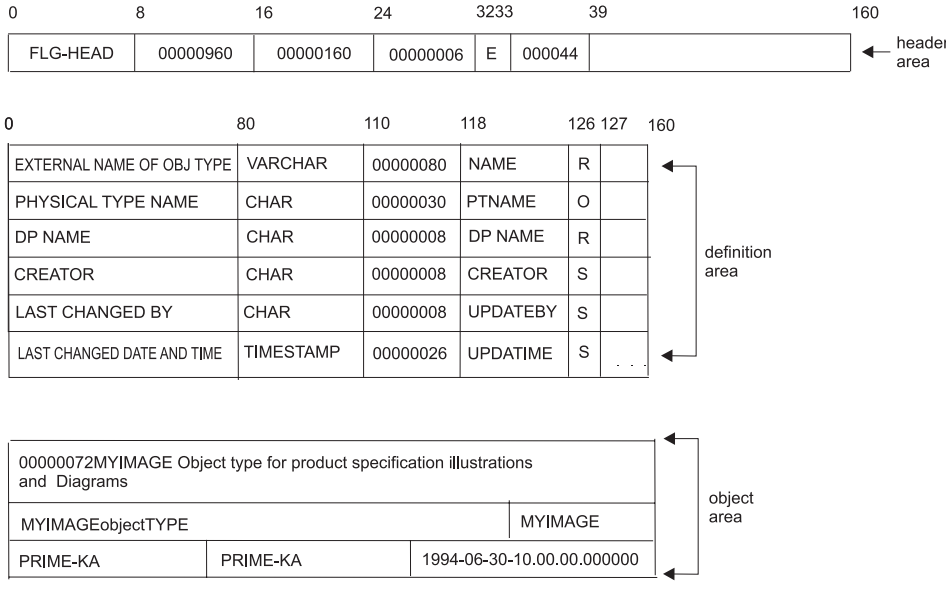

Figure 159. Sample input structure for FLGUpdateReg

In this example, the values in the object areas corresponding to system-generated properties (CREATOR, LAST CHANGED BY, and LAST

CHANGED DATE AND TIME) *cannot* be updated and are ignored by FLGUpdateReg. One way of generating the input structure is to issue FLGGetReg to get the current definition and values and use the output structure from that API call as a template for this FLGUpdateReg input structure.

Bytes 33 through 38 of the header area contain the object type ID (000044) of the object type for which registration information is being updated.

FLGWhereUsed

Retrieves a list of Grouping object instances that contain a specific object instance.

Authorization

Administrator or user

Syntax

```
APIRET  APIENTRY   FLGWhereUsed( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

Parameters

pszFLGID (PSZ) — input
Points to the system-generated unique ID for the contained instance (16 characters).

Characters 1-6 of this ID identify the object type of this instance.

Characters 7-16 of this ID are the system-generated unique instance identifier.

ppListStruct (PFLGHEADERAREA) — output
Points to the address of the pointer to the output structure listing the container objects.

The output structure includes some property specifications and the property values of the container objects. Each container object has the following information:
- FLGID (16 characters)
- Name (80 characters)

All instances are first sorted by object type name, and then sorted by Name; the actual order of the instances depends on the collating sequence used by the database management system for the information catalog.

The maximum number of object instances that can be returned by FLGWhereUsed is approximately 5000, depending on the storage available on your machine.

When there is no output structure, the pointer to the structure is set to NULL.

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Output structure

FLGWhereUsed produces an output structure containing a list of objects that contain the specified object, as shown in Figure 160.

The object area of the output structure contains a list of all the Grouping objects that contain the specified object instance. The returned object instances are identified by the values of the FLGID and object instance external name.

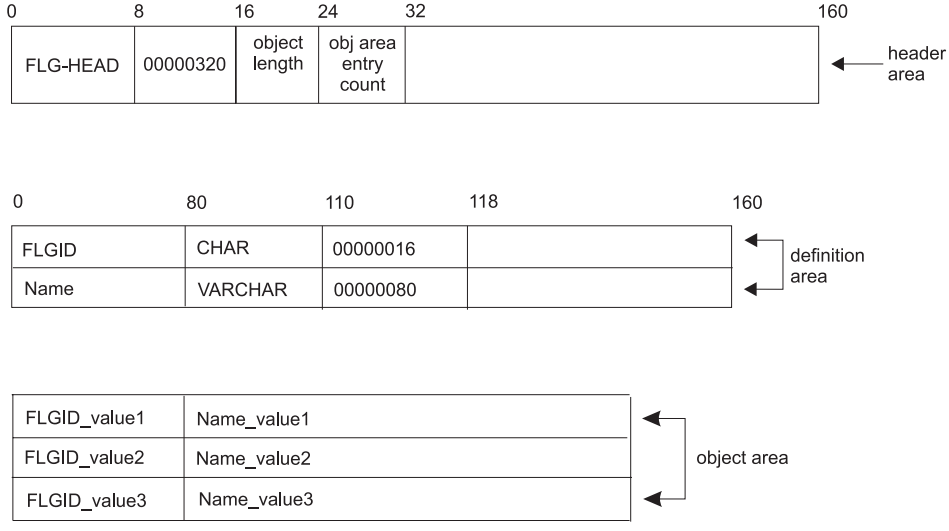

Figure 160. FLGWhereUsed output structure

Usage

Freeing memory allocated for an output structure

If FLGWhereUsed returned data in the output structure, you must save the data returned in the output structure and then call FLGFreeMem (see "FLGFreeMem" on page 125). Do not use other methods, for example, C language instructions, to free memory.

Examples

Figure 161 on page 240 shows the C language code required to issue the FLGWhereUsed API call. This sample code issues the FLGWhereUsed API call.

```
APIRET           rc;                // reason code from FLGWhereUsed
UCHAR            pszFLGID[FLG_ID_LEN + 1];    // Information Catalog Manager ID
PFLGHEADERAREA * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE       ExtCode=0;              // extended code
  .
  .  /* provide values for input parameters */
  .
strcpy (pszFLGID, "0000770000003333");
rc = FLGWhereUsed (pszFLGID,
                   ppListStruct, // address of output structure pointer
                   &ExtCode);
```

Figure 161. Sample C language call to FLGWhereUsed

Figure 162 shows the resulting output structure.

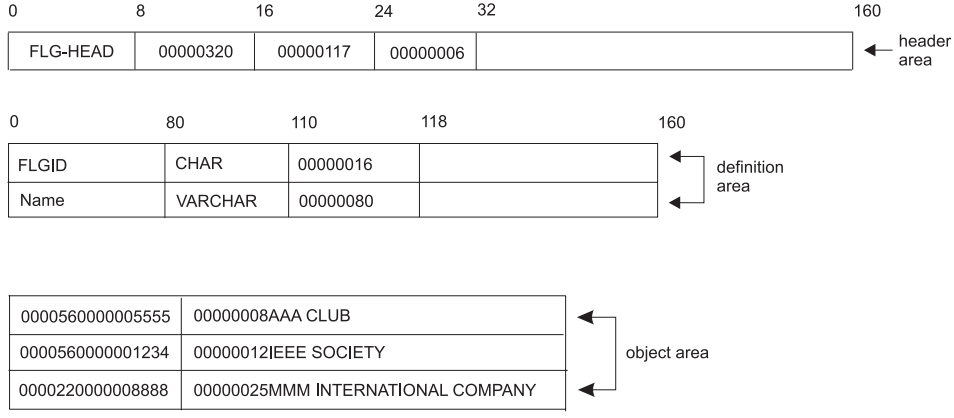

Figure 162. Sample output structure for FLGWhereUsed

The specified object instance is contained by three instances in two different object types. The object type name for the object type ID 000056 is alphabetically less than the object type name for the object ID 000022, and therefore appears first.

FLGXferTagBuf

Transfers the delete history, which is a log of delete activity, to a tag file to duplicate the deletions in other information catalogs, for example, "shadow" information catalogs in a distributed environment.

Authorization

Administrator

Syntax

```
APIRET  APIENTRY   FLGXferTagBuf( PSZ          pszTagFileID,
                                  FLGOPTIONS   Options,
                                  PFLGEXTCODE  pExtCode );
```

Parameters

pszTagFileID (PSZ) — input
Points to the name of the output tag language file. This parameter is required.

For OS/2, this parameter contains the drive, directory path, and file name, and must be valid for a file allocation table (FAT) or HPFS file. The file name and extension (excluding the drive and directories) cannot exceed 240 characters.

The target drive for this file can be either a fixed or removable disk.

Options (FLGOPTIONS) — input
Choose one of the following options for the file to which you want to transfer the delete history:
FLG_TAGOPT_NEW
Create a new file
FLG_TAGOPT_REPLACE
Replaces an existing file

pExtCode (PFLGEXTCODE) — output
Points to an extended code associated with the reason code. See "Appendix D. Information Catalog Manager reason codes" on page 263 to see if a meaningful extended code is associated with the returned reason code.

Reason code (APIRET)
Represents the execution result of this API call.

See "Appendix D. Information Catalog Manager reason codes" on page 263 for an explanation of the returned reason codes.

Usage

FLGXferTagBuf terminates abnormally when the target disk is full, even if the disk is removable.

To protect against erroneous deletions in other information catalogs, you should examine the contents of a delete history tag file before importing it to any other information catalog, especially if you have deleted Grouping object instances, or object types.

Examples

Figure 163 shows the C language code required to issue the FLGXferTagBuf call. This sample code creates the file c:\sampdel.tag, to which it then transfers the delete history.

```
APIRET           rc;               // reason code from API
PSZ              pszTagFile = "c:\\sampdel.tag";
FLGEXTCODE       xc=0;             // extended code
FLGOPTIONS       Options=0;
  .
  .  /*                                                 */
  .
Options=Options | FLG_TAGOPT_NEW;
rc = FLGXferTagBuf (pszTagFile,
                    Options,
                    &xc);
```

Figure 163. Sample C language call to FLGXferTagBuf

Appendix A. Sample program DG2SAMP.C

The Information Catalog Manager provides a sample program, DG2SAMP.C, that you can compile, link, and run. DG2SAMP.C is in your \SQLLIB\LIB directory on the drive where DB2 UDB is installed. This sample program lets the user change the name of an object instance by:

1. Getting a list of the object types in your information catalog.
2. Finding the object you are looking for if it exists.
3. Getting information about the instance.
4. Updating the value of the Name property.

This program issues the following API calls:

- FLGCommit
- FLGFreeMem
- FLGGetInst
- FLGInit
- FLGListObjTypes
- FLGRollback
- FLGSearch
- FLGTerm
- FLGTrace
- FLGUpdateInst

Compiling DG2SAMP.C

To compile DG2SAMP.C using Microsoft Visual C++ Compiler you need to issue the following command while in the same directory as DG2SAMP.C:

```
cl /c DG2SAMP.C
```

Linking DG2SAMP.C

To link your Microsoft Visual C++ Compiler program, issue the following command while in the same directory as DG2SAMP.C:

```
link /dll dgwapi.lib dg2samp.obj
```

Executing DG2SAMP.C

This example uses the DGSAMPLE information catalog provided with the Information Catalog Manager, and assumes that you have administrator authorization to this information catalog.

1. Enter the command DG2SAMP.

2. Enter your user ID.
3. Enter your password.
4. Enter the name of the information catalog.

   For this scenario, enter: `DGSAMPLE`
5. Enter the external name of the object type of the object you want to change.

   For this scenario, enter: `Business groupings`
6. Enter the external name of the object you want to change.

   For this scenario, enter: `Billings`
7. Enter the new external name of the object.

   For this scenario, enter: `Account payment histories`

Appendix B. The Information Catalog Manager API header file—DG2APIH

The Information Catalog Manager provides a header file, DG2API.H that defines the function prototypes of API calls, constants, and data types required for C language applications that use the Information Catalog Manager API calls.

DG2API.H is installed in the VWSLIB\LIB directory on the drive where you installed the Information Catalog Manager.

To use the definition types defined in DG2API.H with the Information Catalog Manager for Windows, you need to include in your program the WINDOWS.H header file included with Microsoft Visual C++ Compiler.

Constants defined in DG2API.H

Table 19 contains variables defined for programs that use the Information Catalog Manager API calls to access the Information Catalog Manager functions.

Table 19. Constants defined in DG2API.H

Input or output structure header area constants	Bytes	Defines length of:
FLG_H_IDENT_LEN	8	Structure identifier (FLG-HEAD)
FLG_H_DEFAREA_LEN	8	Definition length
FLG_H_OBJAREA_LEN	8	Object area length
FLG_H_OBJAREAENT_LEN	8	Object area entry count
FLG_H_CATEGORY_LEN	1	Category
FLG_H_OBJTYPID_LEN	6	Object type ID
FLG_H_RESERVED_LEN	121	Reserved area
FLG_HEADER_SIZE	160	Header area
Input or output structure definition area lengths	**Bytes**	**Defines length of:**
FLG_D_PROPNM_LEN	80	Property name
FLG_D_DATATYP_LEN	30	Data type value
FLG_D_DATA_LEN	8	Data length value
FLG_D_PPN_LEN	8	Property short name
FLG_D_VF_LEN	1	Value flag
FLG_D_US_LEN	1	UUI sequence number
FLG_D_CS_LEN	1	Case- sensitivity flag

Table 19. Constants defined in DG2API.H (continued)

FLG_D_FS_LEN	1	Fuzzy-search flag
FLG_D_RESERVED_LEN	30	Reserved area
FLG_DEFINITION_SIZE	160	Definition area record
Information Catalog Manager string lengths	**Byte length**	**Defines length of:**
FLG_OBJTYPID_LEN	6	Object type ID
FLG_INSTIDNT_LEN	10	Instance ID
FLG_INST_NAME_LEN	80	Instance name
FLG_UPDATIME_LEN	26	Time stamp for when the object type is created or updated
FLG_UPDATEBY_LEN	8	User ID of the person who performed the update
FLG_ID_LEN	16	FLGID value
FLG_EXTERNAL_NAME_LEN	80	Object type external name
FLG_PTNAME_LEN	30	Object type physical type name
FLG_DPNAME_LEN	8	Object type short name
FLG_CREATOR_LEN	8	User ID of the creator of the object type
FLG_USERID_LEN	8	Log on user ID
FLG_PASSWORD_LEN	8	Log on password
FLG_DATABASENAME_LEN	8	Name of the Information Catalog Manager database
FLG_VARIABLE_DATA_LENGTH_LEN	8	Length field for VARCHAR and LONG VARCHAR values
Data type maximum lengths	**Bytes**	**Defines maximum length for:**
FLG_CHAR_MAXLEN	254	CHAR data type
FLG_VARCHAR_MAXLEN	4000	VARCHAR data type
FLG_LONG_VARCHAR_MAXLEN	32700	LONG VARCHAR data type
FLG_TIMESTAMP_MAXLEN	26	TIMESTAMP data type
Maximum values	**Value**	**Defines maximum for:**
FLG_REG_NUM_PROPERTIES	6	Number of registration properties
FLG_ICON_FILE_ID_MAXLEN	259	Length of the path, file name, and extension of the icon file
FLG_TAG_FILE_ID_MAXLEN	259	Length of the path, file name, and extension of the tag language file

Table 19. Constants defined in DG2API.H (continued)

FLG_LOG_FILE_ID_MAXLEN	259	Length of the path, file name, and extension of the log file
FLG_ECHO_FILE_ID_MAXLEN	259	Length of the path, file name, and extension of the echo file
FLG_ICON_PATH_MAXLEN	246	Length of the icon path
FLG_ICON_MAXLEN	30000	Size of the icon file
FLG_UUI_MAXLEN	254	Length in bytes of a UUI property value
FLG_MAX_PROPERTIES	255	Number of properties in an object type
FLG_MAX_NUM_LONG_VARCHARS	14	Number of properties with the LONG VARCHAR data type
FLG_MAXLEN_SEARCH_LONGVARCHAR	3000	Length of search criteria for a property with the LONG VARCHAR data type
FLG_MAX_IMP_EXP_OBJTYPES	3500	Number of unique object types processed in a single tag language file
FLG_MAX_ANCHOR_NUM	1600	Number of object instances returned by FLGListAnchors
FLG_MAX_ORPHAN_NUM	1600	Number of object instances returned by FLGListOrphans
FLG_MAX_CONTAINEE_NUM	1600	Number of object instances returned by FLGFoundIn

Input parameters for FLGConvertID	**Value**	**Description**
FLG_DPNAME	'D'	Object type short name to convert to object type ID
FLG_FLGID	'F'	Object instance ID to convert to extended object instance name

Input options for FLGFoundIn, FLGListAssociates, and FLGListOrphans	**Value**	**Description**
FLG_LIST_PROGRAM	0x00000001	Retrieve Program object instances associated with specified object type
FLG_LIST_CONTAIN	0x00000002	Retrieve object instances contained by a specified instance
FLG_LIST_CONTACT	0x00000003	Retrieve Contact object instances associated with specified instance

Table 19. Constants defined in DG2API.H (continued)

FLG_LIST_ATTACHMENT	0x00000004	Retrieve object instances attached to specified instance
FLG_LIST_COMMENTS	0x00000005	Retrieve Comments object instances attached to specified instance
FLG_LIST_LINK	0x00000006	Retrieve object instances linked to specified instance
Platform options for FLGManageIcons	**Length**	**Description**
FLG_PLATFORM_OS2	0x00000100	OS/2 icons
FLG_PLATFORM_WINDOWS	0x00000200	Windows icons
Output options for FLGManageIcons	**Length**	**Description**
FLG_ICON_EXIST	0x00000001	Specified icon exists
FLG_ICON_NOTEXIST	0x00000002	Specified icon does not exist
Options of FLGRelation	**Value**	**Description**
FLG_CREATE_RELATION	'C'	Create option
FLG_DELETE_RELATION	'D'	Delete option
Types of relationships defined by FLGRelation	**Value**	**Description**
FLG_ATTACHMENT_RELATION	'A'	Attachment relationship
FLG_CONTAINER_RELATION	'C'	Contains relationship
FLG_CONTACT_RELATION	'T'	Contact relationship
FLG_LINK_RELATION	'L'	Link relationship
File options for FLGXferTagBuf	**Value**	**Description**
FLG_TAGOPT_NEW	0x00000001	Create a new file into which to transfer delete history
FLG_TAGOPT_REPLACE	0x00000002	Transfer delete history into an existing file, replacing former contents
Input/output options for FLGManageTagBuf	**Value**	**Description**
FLG_TAGBUF_RESET	0x00000001	Remove existing entries from delete history log
FLG_TAGBUF_QUERY	0x00000002	Query whether delete history log contains entries
FLG_TAGBUF_EMPTY	0x00000001	Delete history log contains no entries
FLG_TAGBUF_NOT_EMPTY	0x00000002	Delete history log contains entries
Delete options for FLGDeleteTree	**Value**	**Description**

Table 19. Constants defined in DG2API.H (continued)

FLG_DELTREE_REL	0x00000001	Delete Grouping object instance and its underlying tree structure
FLG_DELTREE_ALL	0x00000002	Delete Grouping object instance and its underlying tree structure, including underlying objects

FLGManageCommentStatus output structure property short names	**Value**	**Description**
FLG_COMMENT_STATUS1_PPN	"CSTATUS1"	First available status choice for comments
FLG_COMMENT_STATUS2_PPN	"CSTATUS2"	Second available status choice for comments
FLG_COMMENT_STATUS3_PPN	"CSTATUS3"	Third available status choice for comments
FLG_COMMENT_STATUS4_PPN	"CSTATUS4"	Fourth available status choice for comments
FLG_COMMENT_STATUS5_PPN	"CSTATUS5"	Fifth available status choice for comments
FLG_COMMENT_STATUS6_PPN	"CSTATUS6"	Sixth available status choice for comments
FLG_COMMENT_STATUS7_PPN	"CSTATUS7"	Seventh available status choice for comments
FLG_COMMENT_STATUS8_PPN	"CSTATUS8"	Eighth available status choice for comments
FLG_COMMENT_STATUS9_PPN	"CSTATUS9"	Ninth available status choice for comments
FLG_COMMENT_STATUSA_PPN	"CSTATUSA"	Tenth available status choice for comments

User type options for FLGManageUsers	**Value**	**Description**
FLG_USERTYPE_PADMIN	'A'	Primary administrator
FLG_USERTYPE_BADMIN	'B'	Backup administrator
FLG_USERTYPE_POWERUSER	'D'	User authorized to perform additional object management tasks
FLG_USERTYPE_USER	'W'	User

Trace level options set with FLGTrace	**Value**	**Description**
FLG_TRACE_ON	1	Turn tracing on
FLG_TRACE_OFF	0	Turn tracing off
FLG_TRACELEVEL_0	0	Default trace level
FLG_TRACELEVEL_1	1	Include function entry and exit records

Table 19. Constants defined in DG2API.H (continued)

FLG_TRACELEVEL_2	2	Include function-level information
FLG_TRACELEVEL_3	3	Include input and output parameters
FLG_TRACELEVEL_4	4	Include all input and output structures passed to or used by the Information Catalog Manager
Actions performed by the Information Catalog Manager	**Value**	**Description**
FLG_ACTION_CREATE	0x00000001	Creates; for example, adds an icon to an object type or user to an information catalog
FLG_ACTION_DELETE	0x00000002	Deletes; for example, deletes an icon from an object type
FLG_ACTION_UPDATE	0x00000004	Updates; for example, changes the list of available status choices for comments, or toggles on or off recording of delete history
FLG_ACTION_GET	0x00000008	Retrieve current setting or list
FLG_ACTION_QUERY	0x00000010	Determine existence
FLG_ACTION-LIST	0x00000020	Retrieve list of users
Category types	**Value**	**Description**
FLG_GROUPING_OBJ	'G'	Grouping category
FLG_ELEMENTAL_OBJ	'E'	Elemental category
FLG_CONTACT_OBJ	'C'	Contact category
FLG_DICTIONARY_OBJ	'D'	Dictionary category
FLG_PROGRAM_OBJ	'P'	Program category
FLG_SUPPORT_OBJ	'S'	Support category
FLG_ATTACHMENT_OBJ	'A'	Attachment category
Yes and no values	**Value**	**Description**
FLG_YES	'Y'	Yes
FLG_NO	'N'	No
Value flags	**Value**	**Description**
FLG_REQUIRED	'R'	Required property
FLG_OPTIONAL	'O'	Optional property
FLG_SYSTEM	'S'	System- generated property
Universal unique identifier sequence numbers	**Value**	**Description**

Table 19. Constants defined in DG2API.H (continued)

FLG_UUI_1	'1'	First property in UUI
FLG_UUI_2	'2'	Second property in UUI
FLG_UUI_3	'3'	Third property in UUI
FLG_UUI_4	'4'	Fourth property in UUI
FLG_UUI_5	'5'	Fifth property in UUI
FLG_BLANK	' '	A single blank character

Property short names of required properties	**Value**	**Description**
FLG_PPN_OBJTYPID	"OBJTYPID"	Object type ID
FLG_PPN_INSTIDNT	"INSTIDNT"	Instance ID
FLG_PPN_INST_NAME	"NAME"	Name of object instance
FLG_PPN_UPDATIME	"UPDATIME"	Time stamp of date and time last updated
FLG_PPN_UPDATEBY	"UPDATEBY"	User ID of person who performed the last update
FLG_PPN_EXTERNAL_NAME	"NAME"	External name of object type
FLG_PPN_PTNAME	"PTNAME"	Physical type name of object type
FLG_PPN_DPNAME	"DPNAME"	DP NAME (short name) of the object type
FLG_PPN_CREATOR	"CREATOR"	User ID of the person who created the object type

Common property short names—Information Catalog Manager-defined object types	**Value**	**Description**
FLG_PPN_UUICLASS	"UUICLASS"	Defines object class
FLG_PPN_UUIQUAL1	"UUIQUAL1"	First qualifier property used to ensure that the identifier UUI value is unique within the UUI class.
FLG_PPN_UUIQUAL2	"UUIQUAL2"	Second qualifier property used to ensure that the identifier UUI value is unique within the UUI class.
FLG_PPN_UUIQUAL3	"UUIQUAL3"	Third qualifier property used to ensure that the identifier UUI value is unique within the UUI class.
FLG_PPN_UUIDENT	"UUIDENT"	Unique object instance-level identifier used as part of the UUI.
FLG_PPN_HANDLES	"HANDLES"	Identifies object type handled by the program association.
FLG_PPN_STARTCMD	"STARTCMD"	Command to invoke program associated with an object type

Table 19. Constants defined in DG2API.H (continued)

FLG_PPN_PARMLIST	"PARMLIST"	Parameter list to pass to program upon invocation
FLG_PPN_SHRTDESC	"SHRTDESC"	Short description for object instances
FLG_PPN_CREATSTP	"CREATSTP"	Creation date timestamp for Comments objects
FLG_PPN_STATUS	"STATUS"	Status of Comments object instances
FLG_PPN_ACTIONS	"ACTIONS"	Actions to be taken against an object instance, for example, starting a program
FLG_PPN_EXTRA	"EXTRA"	Reserved
FLG_PPN_LONGDESC	"LONGDESC"	Long description of an object instance
FLGImport restart options	**Value**	**Description**
FLG_RESTART_BEGIN	'B'	Import the tag language file from the beginning
FLG_RESTART_CHECKPT	'C'	Import the tag language file starting at the last committed checkpoint.
Export input structure property names	**Value**	**Description**
FLG_EXPORT_DEFAREA_FLGID	"FLGID"	Property passing the FLGID of the object to be exported
FLG_EXPORT_DEFAREA_ATTACHMENT_IND	"ATTACHMENT-IND"	Property indicating whether to export associated Attachment objects
FLG_EXPORT_DEFAREA_CONTAINEE_IND	"CONTAINEE-IND"	Property indicating whether to export contained objects
FLG_EXPORT_DEFAREA_CONTACT_IND	"CONTACT-IND"	Property indicating whether to export associated Contact objects
FLG_EXPORT_DEFAREA_LINK_IND	"LINK-IND"	Property indicating whether to export linked objects
Data types	**Value**	**Description**
FLG_DTYPE_CHAR	"CHAR"	Fixed-length character
FLG_DTYPE_VARCHAR	"VARCHAR"	Variable length character
FLG_DTYPE_LONGVARCHAR	"LONG VARCHAR"	Long variable length character
FLG_DTYPE_TIMESTAMP	"TIMESTAMP"	Time stamp
Header area structure identifier	**Value**	**Description**

Table 19. Constants defined in DG2API.H (continued)

FLG_H_IDENT	"FLG-HEAD"	First value in an Information Catalog Manager input or output structure

Database platform identifiers	Value	Description
FLG_DG2_DB22	0	The Information Catalog Manager using DB2 UDB for OS/2
FLG_DG2_DB2	1	The Information Catalog Manager using DB2 UDB for OS/390® or DB2 for OS/390
FLG_DG2MVS	2	Reserved
FLG_DG2_DB400	4	The Information Catalog Manager using DB2 UDB for AS/400
FLG_DG2_DB6000	6	The Information Catalog Manager using DB2 UDB for AIX
FLG_DG2_DB26000PE	7	The Information Catalog Manager using DB2 PE for AIX or DB2 UDB EEE
FLG_DG2_DB2NT	8	The Information Catalog Manager using DB2 UDB for Windows NT
FLG_DG2_DB295	9	The Information Catalog Manager using DB2 UDB for Windows 95

Structure and data type definitions in DG2API.H

Table 20 and Table 21 on page 254 contain definitions for structures and data types used with the Information Catalog Manager API calls.

Table 20. Structure definitions

Header area	Description

Table 20. Structure definitions (continued)

`typedef struct _FLG_HEADER_AREA {`
`UCHAR pchHIdent [ FLG_H_IDENT_LEN ];`
`UCHAR pchHDefLength [ FLG_H_DEFAREA_LEN ];`
`UCHAR pchHObjLength [ FLG_H_OBJAREA_LEN ];`
`UCHAR pchHObjEntryCount [ FLG_H_OBJAREAENT_LEN ];`
`UCHAR pchHCategory [ FLG_H_CATEGORY_LEN ];`
`UCHAR pchHObjTypeId [ FLG_H_OBJTYPID_LEN ];`
`UCHAR pchHReserved [ FLG_H_RESERVED_LEN ];`
`} FLGHEADERAREA;`
`#ifdef WINDOWS`
`typedef FLGHEADERAREA __huge *PFLGHEADERAREA;`
`#else`
`typedef FLGHEADERAREA *PFLGHEADERAREA;`
`#endif`
	Defines a structure containing all the elements of the header area for an Information Catalog Manager input or output structure

Definition area	**Description**

`typedef struct _FLG_DEFINITION_AREA {`
`UCHAR pchDPropName [ FLG_D_PROPNM_LEN ];`
`UCHAR pchDDataType [ FLG_D_DATATYP_LEN ];`
`UCHAR pchDDataLength [ FLG_D_DATA_LEN ];`
`UCHAR pchDTagName [ FLG_D_PPN_LEN ];`
`UCHAR pchDVF [ FLG_D_VF_LEN ];`
`UCHAR pchDUS [ FLG_D_US_LEN ];`
`UCHAR pchDCS [ FLG_D_CS_LEN ];`
`UCHAR pchDFS [ FLG_D_FS_LEN ];`
`UCHAR pchDReserved [ FLG_D_RESERVED_LEN ];`
`} FLGDEFINITIONAREA;`
`#ifdef WINDOWS`
`typedef FLGDEFINITIONAREA __huge *PFLGDEFINITIONAREA;`
`#else`
`typedef FLGDEFINITIONAREA *PFLGDEFINITIONAREA;`
`#endif`
	Defines a structure containing all the elements of a definition area record for an Information Catalog Manager input or output structure

Table 21. Data type definitions

Synonyms for data types	Data types
FLGRELOPTION	UCHAR—unsigned character
FLGRELTYPE	UCHAR—unsigned character
FLGTRACEOPTION	ULONG—unsigned long integer
FLGIDLENGTH	ULONG—unsigned long integer
FLGOPTIONS	ULONG—unsigned long integer
PFLGOPTIONS	* FLGOPTIONS—pointer to unsigned long integer
FLGADMIN	UCHAR—unsigned character
FLGRESTARTOPTION	UCHAR—unsigned character
FLGEXTCODE	LONG—long integer
PFLGEXTCODE	* FLGEXTCODE—pointer to long integer

Information Catalog Manager API call function prototypes

Table 22 defines the function prototypes for the Information Catalog Manager API calls.

Table 22. API call function prototypes

FLGAppendType

```
APIRET  APIENTRY    FLGAppendType( PFLGHEADERAREA   pObjTypeStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGCommit

```
APIRET  APIENTRY   FLGCommit( PFLGEXTCODE      pExtCode );
```

FLGConvertID

```
APIRET  APIENTRY  FLGConvertID( PSZ              pszInBuffer,
                                PSZ              pszOutBuffer,
                                FLGOPTIONS       Options,
                                PFLGEXTCODE      pExtCode );
```

FLGCreateInst

```
APIRET  APIENTRY    FLGCreateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PSZ              pszFLGID,
                                   PFLGEXTCODE      pExtCode );
```

FLGCreateReg

```
APIRET  APIENTRY  FLGCreateReg( PFLGHEADERAREA   pObjRegStruct,
                                PSZ              pszIconFileID,
                                PSZ              pszObjTypeID,
                                PFLGEXTCODE      pExtCode );
```

FLGCreateType

```
APIRET  APIENTRY   FLGCreateType( PFLGHEADERAREA   pObjTypeStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGDeleteInst

```
APIRET  APIENTRY    FLGDeleteInst( PSZ              pszFLGID,
                                   PFLGEXTCODE      pExtCode );
```

FLGDeleteReg

```
APIRET  APIENTRY  FLGDeleteReg( PSZ            pszObjTypeID,
                                PFLGEXTCODE    pExtCode );
```

FLGDeleteTree

```
APIRET  APIENTRY   FLGDeleteTree( PSZ              pszFLGID,
                                  FLGOPTIONS       Options,
                                  PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGDeleteType

Table 22. API call function prototypes (continued)

```
APIRET  APIENTRY   FLGDeleteType( PSZ              pszObjTypeID,
                                  PFLGEXTCODE      pExtCode );
```

FLGDeleteTypeExt

```
APIRET  APIENTRY   FLGDeleteTypeExt( PSZ              pszObjTypeID,
                                     PFLGEXTCODE      pExtCode );
```

FLGExport

```
APIRET  APIENTRY   FLGExport( PSZ              pszTagFileID,
                              PSZ              pszLogFileID,
                              PSZ              pszIcoPath,
                              PFLGHEADERAREA   pListStruct,
                              PFLGEXTCODE      pExtCode );
```

FLGFoundIn

```
APIRET  APIENTRY   FLGFoundIn( PSZ              pszFLGID,
                               FLGOPTIONS       Options,
                               PFLGHEADERAREA * ppListStruct,
                               PFLGEXTCODE      pExtCode );
```

FLGFreeMem

```
APIRET  APIENTRY    FLGFreeMem( PFLGHEADERAREA   pFLGOutputStruct,
                                PFLGEXTCODE      pExtCode );
```

FLGGetInst

```
APIRET  APIENTRY     FLGGetInst( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppObjInstStruct,
                                 PFLGEXTCODE      pExtCode );
```

FLGGetReg

```
APIRET  APIENTRY   FLGGetReg( PSZ              pszObjTypeID,
                              PSZ              pszIconFileID,
                              PFLGHEADERAREA * ppObjRegStruct,
                              PFLGEXTCODE      pExtCode );
```

FLGGetType

```
APIRET  APIENTRY    FLGGetType( PSZ              pszObjTypeID,
                                PFLGHEADERAREA * ppObjTypeStruct,
                                PFLGEXTCODE      pExtCode );
```

FLGImport

```
APIRET  APIENTRY   FLGImport( PSZ              pszTagFileID,
                              PSZ              pszLogFileID,
                              PSZ              pszIcoPath,
                              FLGRESTARTOPTION RestartOpt,
                              PFLGEXTCODE      pExtCode );
```

FLGInit

Table 22. API call function prototypes (continued)

```
APIRET  APIENTRY   FLGInit( PSZ              pszUserID,
                            PSZ              pszPassword,
                            PSZ              pszDatabaseName,
                            FLGADMIN         Admin,
                            PFLGHEADERAREA * ppListStruct,
                            PFLGEXTCODE      pExtCode );
```

FLGListAnchors

```
APIRET  APIENTRY   FLGListAnchors( PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGListAssociates

```
APIRET  APIENTRY   FLGListAssociates( PSZ              pszInBuffer,
                                      FLGOPTIONS       Options,
                                      PFLGHEADERAREA * ppListStruct,
                                      PFLGEXTCODE      pExtCode );
```

FLGListContacts

```
APIRET  APIENTRY  FLGListContacts( PSZ              pszFLGID,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGListObjTypes

```
APIRET  APIENTRY  FLGListObjTypes( PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGListOrphans

```
APIRET  APIENTRY   FLGListOrphans( PSZ              pszObjTypeID,
                                   FLGOPTIONS       Options,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGListPrograms

```
APIRET  APIENTRY  FLGListPrograms( PSZ              pszObjTypeID,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGManageCommentStatus

```
APIRET  APIENTRY   FLGManageCommentStatus( FLGOPTIONS       Action,
                                           FLGHEADERAREA  * pStatusStruct,
                                           PFLGHEADERAREA * ppStatusStruct,
                                           PFLGEXTCODE      pExtCode );
```

FLGManageFlags

```
APIRET  APIENTRY   FLGManageFlags( FLGOPTIONS   Action,
                                   FLGOPTIONS   FlagType,
                                   UCHAR        chValue,
                                   UCHAR      * pchValue,
                                   PFLGEXTCODE  pExtCode );
```

Table 22. API call function prototypes (continued)

FLGManageIcons

```
APIRET  APIENTRY   FLGManageIcons( PSZ          pszObjTypeID,
                                   PSZ          pszIconFileID,
                                   FLGOPTIONS   InOptions,
                                   PFLGOPTIONS  pOutOptions,
                                   PFLGEXTCODE  pExtCode );
```

FLGManageTagBuf

```
APIRET  APIENTRY   FLGManageTagBuf( FLGOPTIONS       InOptions,
                                    PFLGOPTIONS      pOutOptions,
                                    PFLGEXTCODE      pExtCode );
```

FLGManageUsers

```
APIRET  APIENTRY   FLGManageUsers( FLGOPTIONS       Options,
                                   PFLGHEADERAREA   pListStruct,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGMdisExport

```
APIRET  APIENTRY   FLGMdisExport( PSZ          pszTagFileName,
                                  PSZ          pszLogFileName,
                                  PSZ          pszObjTypeName,
                                  PSZ          pszObjectName,
                                  PFLGEXTCODE  pExtCode );
```

FLGMdisImport

```
APIRET  APIENTRY   FLGMdisImport( PSZ              pszTagFileID,
                                  PSZ              pszLogFileID,
                                  PFLGEXTCODE      pExtCode );
```

FLGNavigate

```
APIRET  APIENTRY    FLGNavigate( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

FLGOpen

```
APIRET  APIENTRY   FLGOpen( PSZ              pszPgmFLGID,
                            PSZ              pszObjFLGID,
                            PFLGEXTCODE      pExtCode );
```

FLGRelation

```
APIRET  APIENTRY   FLGRelation( PSZ              pszSrcFLGID,
                                PSZ              pszTrgFLGID,
                                FLGRELTYPE       RelType,
                                FLGRELOPTION     RelOpt,
                                PFLGEXTCODE      pExtCode );
```

FLGRollback

Table 22. API call function prototypes (continued)

```
APIRET  APIENTRY    FLGRollback( PFLGEXTCODE      pExtCode );
```

FLGSearch

```
APIRET  APIENTRY    FLGSearch( PSZ              pszObjTypeID,
                               PFLGHEADERAREA   pSelCriteriaStruct,
                               PFLGHEADERAREA * ppListStruct,
                               PFLGEXTCODE      pExtCode );
```

FLGSearchAll

```
APIRET  APIENTRY   FLGSearchAll( PFLGHEADERAREA   pSelCriteriaStruct,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

FLGTerm

```
APIRET  APIENTRY   FLGTerm( PFLGEXTCODE   pExtCode );
```

FLGTrace

```
APIRET  APIENTRY   FLGTrace( FLGTRACEOPTION   TraceOpt,
                             PFLGEXTCODE      pExtCode );
```

FLGUpdateInst

```
APIRET  APIENTRY    FLGUpdateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGUpdateReg

```
APIRET  APIENTRY   FLGUpdateReg( PFLGHEADERAREA   pObjRegStruct,
                                 PSZ              pszIconFileID,
                                 PFLGEXTCODE      pExtCode );
```

FLGWhereUsed

```
APIRET  APIENTRY   FLGWhereUsed( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

FLGXferTagBuf

```
APIRET  APIENTRY   FLGXferTagBuf( PSZ          pszTagFileID,
                                  FLGOPTIONS   Options,
                                  PFLGEXTCODE  pExtCode );
```

Appendix C. Information Catalog Manager limits

Table 23 describes certain Information Catalog Manager limits.

Table 23. The Information Catalog Manager limits

Information Catalog Manager values	Limit
Longest information catalog database name	30 characters
Longest information catalog physical table name (PT NAME)	30 characters
Longest physical table name (PT NAME) with DB2 UDB for OS/2	18 characters
Longest physical table name (PT NAME) with DB2 UDB for OS/390	18 characters
Longest UUI property value length	254 bytes
Maximum for total of five UUI property value lengths	1270 bytes
Largest Information Catalog Manager object type icon	30000 bytes
Most properties in an object type	255
Most properties with LONG VARCHAR data type in an Information Catalog Manager object type	14
Longest search criteria length for a LONG VARCHAR property	3000 bytes
Maximum number of unique object types processed with ACTION.OBJTYPE() tags in a single tag language file	3500
Maximum number of objects returned for the following API calls: FLGListAnchors FLGListOrphans	1600
Maximum number of objects returned for the following API calls: FLGFoundIn FLGListAssociates FLGListContacts FLGListPrograms FLGNavigate FLGSearch FLGSearchAll FLGWhereUsed	5000

Appendix D. Information Catalog Manager reason codes

Table 24 contains all the reason codes produced by the Information Catalog Manager. The reason codes are ordered by number, and include the mnemonic name, the extended code, and an explanation of what condition produces the reason code.

Certain reason codes produce extended codes, which provide more information about the error situation. If a reason code returns an extended code, the possible meanings of the extended code are listed.

Table 24. Information Catalog Manager reason codes

Number	Reason code	Extended codes	Explanation
0	FLG_OK	—	Completed successfully.
1	FLG_WRN	—	Place holder; indicates the beginning of the numeric range for warnings.
201	FLG_WRN_DISCONNECTED	—	The database has been disconnected.
202	FLG_WRN_DBM_ALREADY_STARTED	—	The database manager was already started before the Information Catalog Manager initialization.
203	FLG_WRN_DB_RESTART	—	The database manager needed to be restarted before the Information Catalog Manager initialization.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
204	FLG_WRN_DB_ACTIVE	—	The specified database manager was already active before the Information Catalog Manager initialization.
1001	FLG_WRN_INST_NOTFOUND	—	Unable to find the object instance (also used by FLGListOrphans, FLGFoundIn, FLGListAssociates, and FLGExport).
1002	FLG_WRN_CONTAINER_NOTFOUND	—	Unable to find a container for the specified object instance.
1003	FLG_WRN_CONTAINEE_NOTFOUND	—	Unable to find any objects contained by the specified object instance.
1004	FLG_WRN_CONTACT_NOTFOUND	—	Unable to find a contact for the specified object instance.
1005	FLG_WRN_PROGRAM_NOTFOUND	—	Unable to find a program associated with this object type.
1006	FLG_WRN_ANCHOR_NOTFOUND	—	Unable to find any anchors (subjects) defined in the Information Catalog Manager database.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
1007	FLG_WRN_PROGRAM_CHANGED	—	One or more associated program instances were changed when the object type was deleted.
1008	FLG_WRN_NO_INPARM_ICON_FILE	—	FLGGetReg API call did not specify a pointer to receive the name of the retrieved icon file. The Information Catalog Manager did not return an icon.
1009	FLG_WRN_NO_ICON	—	No icon associated with the object type.
1010	FLG_WRN_ID_LIMIT_REACHED	—	Reached the maximum number of object types limit.
1011	FLG_WRN_OBJECT_NOT_CHANGED	—	Reserved
1012	FLG_WRN_EXCEED_MAX_ANCHORNUM	Actual number of anchors	Unable to return all anchors (subjects) defined in the Information Catalog Manager database.
1013	FLG_WRN_ICON_REPLACED	—	An icon file already existed in the specified ICOPATH. The icon file was replaced.
1014	FLG_WRN_PROPDUP	—	The property to be appended already exists.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
1015	FLG_WRN_EXCEED_MAX_ORPHANNUM	Actual number of orphans	Exceeded the maximum number of orphans.
1016	FLG_WRN_DB_ICON_REPLACED	—	The object type icon has been replaced in the catalog.
1017	FLG_WRN_LINKOBJ_NOTFOUND	—	Unable to find a linked object for the specified object instance.
1018	FLG_WRN_ATTACHOBJ_NOTFOUND	—	Unable to find attachment objects for the specified object instance.
1019	FLG_WRN_MISSING_PROPS_IN_IOSTRUCT	—	The input structure contains less properties than that defined for the object type. All missing properties are optional. Object instance is created/updated.
2002	FLG_WRN_NO_DISKCNTL_TAG_PRESENTED	—	DISKCNTL is not the first tag in the input tag language file on a removable device. Importing continues, but only the tag language file on the current diskette is processed.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
2003	FLG_WRN_NEED_NEW_TAGFILE_DISKETTE	—	Insert the next diskette to continue importing the tag language file.
2004	FLG_WRN_ICONFILE_OPENERR	—	Reserved
2005	FLG_WRN_NOTHING_TO_IMPORT	—	Unable to find any data to import in the tag language file or in the part of the tag language file after the last checkpoint. The file or part of the file may be empty or contain only COMMENT or DISKCNTL tags.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
2006	FLG_WRN_ICONFILE_RETRIEVE_ERROR	Reason code	API FLGCreateReg or FLGUpdateReg encountered an error while retrieving (opening, reading, or closing) the icon file specified in parameter pszIconFileID. The reason code returned in the extended code indicates the error. FLGCreateReg and FLGUpdateReg have completed all other registration processing successfully.
2007	FLG_WRN_P_HANDLES_CLEARED	—	FLGImport cleared the HANDLES property value for a program instance, because this value refers to an object type that does not exist in the target information catalog.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
2501	FLG_WRN_CFLAG_IGNORED	—	CONTAINEE-IND value for the exported object was ignored because the object does not belong to the Grouping category.
2502	FLG_WRN_TFLAG_IGNORED	—	CONTACT-IND value for the exported object was ignored because the object does not belong to the Grouping or Elemental categories.
2503	FLG_WRN_NO_ICOPATH	—	No icon path was specified; no icons were exported.
2504	FLG_WRN_GETREG_WARNING	Reason code	Export encountered a warning from FLGGetReg. The extended code contains the reason code returned by FLGGetReg.
2505	FLG_WRN_GETINST_WARNING	Reason code	Export encountered a warning from FLGGetInst. The extended code contains the reason code returned by FLGGetInst.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
2506	FLG_WRN_LISTCONTACTS_WARNING	Reason code	Export encountered a warning from FLGListContacts. The extended code contains the reason code returned by FLGListContacts.
2507	FLG_WRN_NAVIGATE_WARNING	Reason code	Export encountered a warning from FLGNavigate. The extended code contains the reason code returned by FLGNavigate.
2508	FLG_WRN_AFLAG_IGNORED	—	ATTACHMENT-IND value for the exported object was ignored because the object is in the Attachment category and cannot have associated attachment objects.
2509	FLG_WRN_LFLAG_IGNORED	—	LINK-IND value for the exported object was ignored because the object does not belong to the Grouping or Elemental categories.
2601	FLG_WRN_NO_HISTORY	—	There is no history entry in the history buffer.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
2602	FLG_WRN_NO_TYPE_RELATE_TO_PROGRAM	—	There is no object type related to the program instance.
7500	FLG_WRN_VIEW_NOT_SUPPORTED	—	View ″T″ is specified in the Tool profile, but this function is not supported by the Information Catalog Manager.
7501	FLG_WRN_LEVEL_NOT_SUPPORTED	—	Level ″T″ is specified in the Tool profile, but this function is not supported by the Information Catalog Manager.
7505	FLG_WRN_NO_BEGIN_DEFINITION_SECTION	—	The BEGIN DEFINITION section is missing from the tag language file.
7510	FLG_WRN_VALUE_TRUNCATED	—	A value is truncated because it exceeded the maximum allowable length.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
7515	FLG_WRN_INV_TIMESTAMP_FORMAT	—	A date or time value does not follow the correct format. Format for date values: YYYY-MM-DD. Format for time values: HH.MM.SS Format for refresh date values: YYYY-MM-DD-HH.MM.SS.
30000	FLG_ERR	—	Place holder; indicates the beginning of the numeric range for errors.
30001	FLG_ERR_INVALID_NUM_STR	—	The numeric string passed to the Information Catalog Manager as input is invalid.
30002	FLG_ERR_INVALID_NUMBER	—	The integer value passed to the Information Catalog Manager as input is too large.
30003	FLG_ERR_BUFF_TOO_SMALL	—	Information Catalog Manager internal error.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
30004	FLG_ERR_MSGFILE_NOTFOUND	—	Unable to locate the Information Catalog Manager message file (DG*xy*MSG.MSG or DG*xy*STR.MSG, where *x* is the platform identifier and *y* is the national language version identifier). This file must be in the Information Catalog Manager working directory.
30005	FLG_ERR_MSGID_NOTFOUND	—	The message identifier could not be located in the message file.
30006	FLG_ERR_CANT_ACCESS_MSGFILE	—	Unable to open the Information Catalog Manager message file.
30007	FLG_ERR_INVALID_MSGFILE_FORMAT	—	The message file (DG*xy*MSG.MSG or DG*xy*STR.MSG, where *x* is the platform identifier and *y* is the national language version identifier) is corrupted or invalid. Reinstall the affected file.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
30008	FLG_ERR_MSGFILE_ERROR	—	Information Catalog Manager internal error.
30009	FLG_ERR_TRACE_FAIL	—	An error occurred in the Information Catalog Manager trace function. The trace file may be corrupted or incomplete.
30010	FLG_ERR_INTERNAL_ERROR	Reason code	The Information Catalog Manager encountered an internal error. Check the reason code returned in the extended code and try to remedy the problem; if this is unsuccessful, call your IBM Service Representative.
30011	FLG_ERR_RESDLL_NOT_LOADED	—	Language DLL file is not found.
30012	FLG_ERR_DGPATH_NOT_FOUND	—	Environment path (DG2PATH) was not set in the CONFIG.SYS file. Environment path (DGWPATH) was not set in either the system registry or the AUTOEXEC.BAT file.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
30013	FLG_ERR_CP_LOAD_FAILED	—	The primary and secondary code pages specified in your CONFIG.SYS file are not supported by the Information Catalog Manager.
30014	FLG_ERR_DBSEM_ERROR	—	Information Catalog Manager internal error (can't get database semaphore).
30015	FLG_ERR_STRINGFILE_ERROR	—	Reserved
30016	FLG_ERR_MSG_TOO_LONG	—	Information Catalog Manager internal error.
30017	FLG_ERR_DG_DB_INUSE	—	User tried to log on to the same Information Catalog Manager database twice.
30018	FLG_ERR_DGLANG_PATH_NOT_FOUND	—	The Information Catalog Manager language dependent directory path cannot be found.
30019	FLG_ERR_INV_DG_CP	—	The code pages specified on the workstation are not supported by the Information Catalog Manager.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
30020	FLG_ERR_INV_DB_CP	—	The code pages specified on the workstation are not supported by the database.
30021	FLG_ERR_VWSPATH_NOT_FOUND	—	Environment path (VWSPATH) was not set in either the system registry or the AUTOEXEC.BAT file.
31000	FLG_ERR_DBERROR	Database SQLCODE	An unexpected database error has occurred. See the database documentation for an explanation of the SQLCODE.
31001	FLG_ERR_DBDISC_FAIL	—	Error occurred while disconnecting from the database.
31002	FLG_ERR_NODBACCESS	—	You cannot access the specified Information Catalog Manager database. Ask the administrator or database administrator for the database authorization you need.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
31003	FLG_ERR_ID_LIMIT_EXCEEDED	—	The system-generated ID (object type ID or instance ID) exceeds the maximum number of IDs allowed in the Information Catalog Manager database. This limit is 99999999 for object instance IDs, and 999999 for object type IDs.
31004	FLG_ERR_PROP_LIMIT_EXCEEDED	—	Exceeded the maximum number of properties (255) allowed for an object type.
31005	FLG_ERR_LONG_VARCHAR_LIMIT_EXCEEDED	Sequence number of property	Exceeded the maximum number of LONG VARCHAR properties (14) allowed for an object type.
31006	FLG_ERR_PTNAME_EXCEEDS_ENVSIZE	—	The physical type name for the object type exceeds the maximum length allowed. This maximum length depends on the underlying database you are using.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
31007	FLG_ERR_DBNAME_NOT_FOUND	—	Unable to find the Information Catalog Manager database. If the database is local, the database name was not found. If the database is remote, the database name was not defined in the local database directory.
31008	FLG_ERR_SRH_CRITERIA_TOOLONG	—	The total length of the search criteria is too long. The maximum length for the sum of the lengths for all specified search criteria is about 32700 bytes, depending on the number of properties in the search criteria.
31009	FLG_ERR_DB_TRANSLOG_FULL	—	The database transaction log is full. Issue FLGCommit or FLGRollback immediately. Increase the database log file size to increase the number of changes possible before you need to commit the changes.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
31010	FLG_ERR_INVALID_AUTHENTICATION	—	The database was cataloged with an incorrect authentication option.
31011	FLG_ERR_CHARCONV_WINTODBM	—	An error occurred while converting a character from the Windows code page to the database code page.
31012	FLG_ERR_DB_TIMEOUT	—	Database server is busy or deadlocked.
31013	FLG_ERR_NOT_SUPPORTED_BY_DB	—	This function is not supported by the database server.
31014	FLG_ERR_DB_ICON_EXIST	—	FLGManageIcons was called with the InOptions parameter set to FLG_ACTION_CREATE, but the icon specified in pszIconFileID already exists in the database. Specify a different icon file, or use FLG_ACTION_UPDATE.
32000	FLG_ERR_REG_NOTEXIST	—	No registration information exists for the specified object type.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32001	FLG_ERR_TYPEID_NOTEXIST	—	No registration information exists for the specified object type.
32002	FLG_ERR_SRCTYPEID_NOTEXIST	—	The specified source object type does not exist.
32003	FLG_ERR_TRGTYPEID_NOTEXIST	—	The specified target object type does not exist.
32004	FLG_ERR_INSTID_NOTEXIST	—	The specified object ID (FLGID) does not exist.
32005	FLG_ERR_SRCINSTID_NOTEXIST	—	The specified source object ID (FLGID) does not exist.
32006	FLG_ERR_TRGINSTID_NOTEXIST	—	The specified target object ID (FLGID) does not exist.
32007	FLG_ERR_PROP_NOTEXIST	—	Unable to start the specified program. The property specified in the program object parameter list is not defined for the object instance.
32008	FLG_ERR_REL_NOTEXIST	—	Unable to delete the relationship because it does not exist.
32009	FLG_ERR_TYPE_NOT_CREATED	—	The specified object type has been registered but not created.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32010	FLG_ERR_SRCTYPE_NOT_CREATED	—	The object type specified in the FLGID of the source object instance has been registered but not created.
32011	FLG_ERR_TRGTYPE_NOT_CREATED	—	The object type specified in the FLGID of the target object instance has been registered but not created.
32012	FLG_ERR_INV_P_CATEGORY	—	P (Program) is an invalid value for the category when creating or deleting object types. You cannot create or delete Program category object types.
32013	FLG_ERR_INV_P_HANDLE_CAT	—	The HANDLES property value of the Program object instance is invalid. The value must be the name of a non-PROGRAM object type.
32014	FLG_ERR_P_HANDLE_NOTEXIST	—	The HANDLES property value of the Program object instance is invalid. The specified object type does not exist.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32015	FLG_ERR_P_HANDLE_NOT_CREATED	—	The HANDLES property value of the Program object instance is invalid. The specified object type has been registered, but not created.
32016	FLG_ERR_INV_A_CATEGORY	—	A (Attachment) is an invalid value for the category when creating, deleting, or appending to object types. You cannot create, delete, or append to Attachment category object types.
32300	FLG_ERR_REG_DUP	—	Unable to register the object type. The specified object type has already been registered.
32301	FLG_ERR_TYPE_DUP	—	Unable to create an object type with the specified name. The specified object type name already exists in the Information Catalog Manager database.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32302	FLG_ERR_INST_DUP	—	Unable to create the specified object instance. The Information Catalog Manager database already contains an object instance with identical UUI property values.
32303	FLG_ERR_REL_DUP	—	Unable to create the specified object relationship. The relationship already exists.
32304	FLG_ERR_REL_RECURSIVE	—	Unable to create the specified relationship. The specified relationship would cause a Grouping object to contain itself.
32305	FLG_ERR_UUI_DUP	Sequence number of property that duplicates the UUI sequence number	The definition of this object type or object contains two or more properties with the same UUI sequence number.
32306	FLG_ERR_INVALID_LINK_RELATION	—	The specified LINK relationship is invalid, because the linker and linkee are the same.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32307	FLG_ERR_INVALID_ATTACHMENT_RELATION	—	The attachment relationship is rejected because the target object is already related to some non-attachment source object. Attachment category objects can be associated to only one non-attachment category source object.
32308	FLG_ERR_ICONFILE_RETRIEVE_ERROR	Reason code	API FLGManageIcons encountered an error while retrieving (opening, reading, or closing) the icon file specified in parameter pszIconFileID. This applies to input options FLG_ACTION_CREATE or FLG_ACTION_UPDATE only. The reason code returned in the extended code indicates the error. Processing is unsuccessful.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32400	FLG_ERR_CONTAINEE_EXIST	—	Unable to delete this object instance because this Grouping object instance contains one or more object instances. You cannot delete this object instance until you delete either the relationships or the contained objects.
32401	FLG_ERR_INST_EXIST	—	Unable to delete the specified object type because instances of the object type exist. You cannot delete this object type until you delete all its instances.
32402	FLG_ERR_TYPE_EXIST	—	Unable to delete the object type registration because its object type exists. You cannot delete this object type registration until the object type is deleted.
32403	FLG_ERR_CONTAINEE_DIFFTYPE	—	FLGDeleteTypeExt API stopped, because it found a containee belonging to a different object type.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32500	FLG_ERR_INVALID_SRCCAT	—	Unable to create the specified relationship. The category for the source object type is invalid.
32501	FLG_ERR_INVALID_TRGCAT	—	Unable to create the specified relationship. The category for the target object type is invalid.
32502	FLG_ERR_INVALID_CAT	—	The category of the input object type is incorrect. Refer to the specific documentation for the API you called for the required input object type.
32600	FLG_ERR_KAEXIST	—	Unable to log on as an administrator. Another administrator is already logged on. The Information Catalog Manager allows only one administrator to log on at a time.
32601	FLG_ERR_NOTAUTH	—	The current user ID is not authorized to use this Information Catalog Manager function.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32602	FLG_ERR_NOT_INITIALIZED	—	The Information Catalog Manager is not initialized. FLGInit must be issued before the Information Catalog Manager can perform any other functions.
32603	FLG_ERR_ALREADY_INITIALIZED	—	The Information Catalog Manager has already been initialized. You cannot issue a second FLGInit call before issuing an FLGTerm call.
32604	FLG_ERR_NOT_CREATOR	—	You do not have the authority to update Comments objects you did not create.
32700	FLG_ERR_INVALID_TYPEID	—	The specified object type ID (OBJTYPID) is invalid.
32701	FLG_ERR_INVALID_TYPEID_LEN	—	The specified object type ID (OBJTYPID) is invalid. This value must be 6 bytes long.
32702	FLG_ERR_INVALID_TYPEID_VAL	—	The value of the specified object type ID (OBJTYPID) is invalid.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32703	FLG_ERR_INVALID_FLGID	Number of exported objects or position of parameter	The specified object ID (FLGID) is invalid.
32704	FLG_ERR_INVALID_FLGID_LEN	—	The object ID (FLGID) is invalid. This value must be 16 bytes long.
32705	FLG_ERR_INVALID_FLGID_VAL	—	The object ID (FLGID) contains invalid characters.
32706	FLG_ERR_INVALID_TYPNM	—	The object type name is invalid.
32707	FLG_ERR_INVALID_INSTNM	—	The object instance name is invalid.
32708	FLG_ERR_INVALID_TIMESTAMP	Sequence number of property	The input value is invalid. The input value must be a time stamp of the form YYYY-MM-DD-HH.MM.SS.NNNNNN and 26 bytes long.
32709	FLG_ERR_INVALID_SRCID	—	The source object ID (FLGID) is invalid.
32710	FLG_ERR_INVALID_TRGID	—	The target object ID (FLGID) is invalid.
32711	FLG_ERR_INVALID_RELTYPE	—	The specified relation type (RelType) is invalid. Valid values are C, T, A, or L.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32712	FLG_ERR_INVALID_RELOPT	—	The specified relation option (RelOpt) is invalid. Valid values are C or D.
32713	FLG_ERR_INVALID_PGM_FLGID	—	The specified object ID (FLGID) for the program object is invalid.
32714	FLG_ERR_INVALID_OBJ_FLGID	—	The specified object ID (FLGID) for the object providing parameters for the FLGOpen call is invalid.
32718	FLG_ERR_INVALID_USERID	—	The user ID value is invalid. The length must be 1-8 characters. User ID/password is invalid (password is case sensitive on AIX®). User is not logged on to the remote node (DB2 for OS/2 V2.1).
32719	FLG_ERR_INVALID_PASSWORD	—	The specified password is invalid. The length must be 1-8 characters.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32720	FLG_ERR_INVALID_DBNAME	—	The specified Information Catalog Manager database name is invalid. The length must be 1-8 characters.
32721	FLG_ERR_INVALID_ADMINOPT	—	The specified user option (admin) is invalid. Valid values are Y and N.
32722	FLG_ERR_INVALID_TRACEOPT	—	The trace option (TraceOpt) is invalid. Valid options are: 0, 1, 2, 3, and 4.
32723	FLG_ERR_NULL_PARAMETER	Position of parameter	A parameter required as input to this API call is missing or null. The extended code indicates the position of the null parameter.
32724	FLG_ERR_NULL_EXTCODE	—	The extended code pointer parameter (pExtCode) is null.
32725	FLG_ERR_INVALID_CONVERTOPT	—	The specified input option (Options) was invalid. Valid values are D, or F.
32726	FLG_ERR_INVALID_ICONOPT	—	The specified input options (Options) are not valid for FLGManageIcons.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32727	FLG_ERR_INVALID_TAGBUFOPT	—	The InOptions specified for FLGManageTagBuf API is not valid. Use FLG_TAGBUF_QUERY or FLG_TAGBUF_RESET as defined in the DGxAPI.H file.
32728	FLG_ERR_INVALID_TAGFILEOPT	—	The Options parameter specified for FLGXferTagBuf API is not valid. Use FLG_TAGOPT_NEW or FLG_TAGOPT_REPLACE as defined in the DGxAPI.H file.
32729	FLG_ERR_INV_DGFLAG_ACTION	—	The Action parameter specified for FLGManageFlags is not valid. Use FLG_ACTION_GET or FLG_ACTION_UPDATE as defined in DGxAPI.H file.
32730	FLG_ERR_INV_DGFLAG_FLAGTYPE	—	The FlagType parameter specified for the FLGManageFlags API is not valid. Use FLG_HISTORY_TYPE_DEI as defined in the DGxAPI.H file.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32731	FLG_ERR_INV_DGFLAG_VALUE	—	The chValue parameter specified for FLGManageFlags is not valid. Valid values are FLG_YES or FLG_NO.
32732	FLG_ERR_INV_STATUS_ACTION	—	The Action parameter specified for the FLGManageCommentStatus API is not valid. Use FLG_ACTION_UPDATE or FLG_ACTION_GET as defined in the DGxAPI.H file.
32733	FLG_ERR_INV_STATUS_LEN	Sequence number of property	The input structure object area contains a status field that is longer than 80 characters.
32734	FLG_ERR_INVALID_TREEOPT	—	The Options parameter specified for FLGDeleteTree API is not valid. Use FLG_DELTREE_REL or FLG_DELTREE_ALL as defined in the DGxAPI.H file.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
32735	FLG_ERR_INVALID_ASSOCOPT	—	The Options parameter specified for FLGListAssociates API is not valid. Use FLG_LIST_PROGRAM, FLG_LIST_ATTACHMENT FLG_LIST_COMMENTS, FLG_LIST_CONTAIN, FLG_LIST_CONTACT or FLG_LIST_LINK as defined in the DGxAPI.H file.
32736	FLG_ERR_INVALID_ORPHANOPT	—	The Options parameter specified for the FLGListOrphans API is not valid. Use FLG_LIST_PROGRAM, FLG_LIST_CONTACT, FLG_LIST_ATTACHMENT or FLG_LIST_COMMENTS as defined in the DGxAPI.H file.
32737	FLG_ERR_INVALID_FOUNDINOPT	—	The Options parameter specified in the FLGFoundIn API is not valid. Use FLG_LIST_PROGRAM, FLG_LIST_CONTAIN, FLG_LIST_CONTACT or FLG_LIST_ATTACHMENT as defined in the DGxAPI.H file.
33000	FLG_ERR_ICON_NOTEXIST	—	The specified icon file does not exist.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34000	FLG_ERR_INVALID_IOSTRUCT	—	The input structure is invalid. Either the definition area length or object area length does not match the length of the area it describes.
34001	FLG_ERR_NO_DEFN_AREA	—	The definition area is missing in the input structure.
34002	FLG_ERR_NO_OBJ_AREA	—	The object area is missing in the input structure.
34003	FLG_ERR_INVALID_POSITION	—	Information Catalog Manager internal error.
34004	FLG_ERR_IOSTRUCT_CONVERSION	—	An Information Catalog Manager internal error occurred while reading the input structure or writing the output structure.
34005	FLG_ERR_INVALID_IOSTRUCT_NULL	Byte offset	The input structure contains a null character.
34006	FLG_ERR_OBJLEN_OBJCNT_MISMATCH	—	Either the object area entry count or the object area length is zero. If one of the values is greater than zero, the other value cannot be zero.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34200	FLG_ERR_INV_HEADER_IDENT	—	The identifier in the input structure header area is invalid. The identifier must be FLG-HEAD.
34201	FLG_ERR_INV_HEADER_DEFLEN	—	The definition length in the input structure header area is not valid. The definition length must be greater than 0 and a multiple of 160. Some API calls require a fixed definition length; see the syntax for the API call for the required definition length.
34202	FLG_ERR_INV_HEADER_DEFCNT	—	The number of definitions expected based on the definition length in the header area is invalid for FLGExport. The number of definitions must be five for FLGExport; therefore, the definition length must be 800.
34203	FLG_ERR_INV_HEADER_OBJLEN	—	The object length in the input structure header area is not valid.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34204	FLG_ERR_INV_HEADER_OBJCNT	—	The object area entry count in the input structure header area is not valid.
34205	FLG_ERR_INV_HEADER_CATEGORY	—	Invalid category specified in header area. For FLGCreateReg, the category value must be one of the following: G, E, C, D, or S. For FLGCreateType, FLGCreateInst, FLGUpdateReg, FLGAppendType, and FLGUpdateInst, the category value must match the value for the related object type registration.
34206	FLG_ERR_INV_HEADER_OBJTYPEID	—	The value of the object type ID in the header area is invalid. This value must be identical to the object type ID generated for the related object type registration.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34207	FLG_ERR_CONFLICTING_HEADER_FIELDS	—	The number of properties derived from the definition length conflicts with the object area entry count in the header area. The number of properties equals the definition area length divided by 160, and the object area entry count must be evenly divisible by the number of properties.
34208	FLG_ERR_CONFLICTING_OBJTYPID	Sequence number of property	The value specified for the object type identifier (OBJTYPID) in the object area does not match the object type ID in the header area.
34209	FLG_ERR_HEADER_DEFLEN_EXCEEDS_MAX	—	The definition length in the header area exceeds the maximum number of properties.
34210	FLG_ERR_NONBLANK_HEADER_CATEGORY	—	The category value in the header area is invalid.
34211	FLG_ERR_NONBLANK_HEADER_OBJTYPEID	—	The object type ID value in the header area is invalid.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34222	FLG_ERR_NONBLANK_HEADER_RESERVED	—	The reserved area of the input structure header area must always be blank.
34500	FLG_ERR_INV_PROPERTY_NAME	Sequence number of property	The specified property name is not one of the property names required with this API call.
34501	FLG_ERR_INV_PROPERTY_PPNAME	Sequence number of property	The property short name for a property in the definition area is invalid. The value may be missing, using DBCS characters, or not using the value required by the API call.
34502	FLG_ERR_INV_PROPERTY_DATATYPE	Sequence number of property	The data type for a property in the definition area is invalid. Valid values are CHAR, TIMESTAMP, VARCHAR, or LONG VARCHAR, depending on the API call.
34503	FLG_ERR_INV_PROPERTY_V_FLAG	Sequence number of property	The value flag for the indicated property in the definition area is invalid. Valid values are R, O, or S.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34504	FLG_ERR_INV_PROPERTY_SVALUE_V_FLAG	Sequence number of property	The value flag for the indicated property in the definition area is invalid. The specified value flag is S, but the Information Catalog Manager does not generate the property indicated by the property short name.
34505	FLG_ERR_INV_PROPERTY_CS_FLAG	Sequence number of property	The case-sensitivity flag value for the indicated property in the definition area is invalid. Valid values are Y or N.
34506	FLG_ERR_INV_PROPERTY_FS_FLAG	Sequence number of property	The fuzzy search flag value for the indicated property in the definition area is invalid. Valid values are Y or N.
34507	FLG_ERR_INV_PROPERTY_UUISEQ	Sequence number of property	The UUI Sequence for the indicated property in the definition area is invalid. Valid values are 1, 2, 3, 4, 5, or blank.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34508	FLG_ERR_INV_PROPERTY_LEN_FOR_DTYPE	Sequence number of property	The length value is invalid for the indicated property in the definition area because of the defined data type.
34509	FLG_ERR_INV_PROP_LEN_FIELD	Sequence number of property	The length for the indicated property in the definition area is invalid. Check the API call syntax for the required length.
34510	FLG_ERR_INV_PROP_VAL_LEN	—	The length field for a VARCHAR or LONG VARCHAR property value in the object area is invalid; it must contain right-aligned numeric characters.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34511	FLG_ERR_INV_RQDPROP_SPEC	Sequence number of property	In a property definition in the definition area, one or more fields required to define a required property are invalid. For a required property, the following fields must be specified as shown in the input structure diagrams for the API call: • Property name (bytes 0-79) • Data type (bytes 80-109) • Length (bytes 110-117) • Property short name (bytes 118-125) • Value flag (byte 126) • UUI sequence number (byte 127)
34512	FLG_ERR_DUP_PROPERTY_NAME	Sequence number of property	Another property in the input structure already has this property name. Each property name must be unique in the input structure.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34513	FLG_ERR_DUP_PROPERTY_PPNAME	Sequence number of property	The property short name for the indicated property is identical to the property short name of another property in this input structure. Each property short name must be unique in the input structure.
34514	FLG_ERR_INV_TOT_UUI_LEN	—	Reserved
34515	FLG_ERR_INV_UUI_LENGTH	UUI sequence number	The indicated UUI property length value in the definition area exceeds the maximum length for a UUI property.
34516	FLG_ERR_MISSING_PROPERTY	—	The definition area for the object instance does not contain all the properties defined for the object type.
34517	FLG_ERR_MISSING_PROPERTY_NAME	Sequence number of property	The property name is required but missing for the indicated property in the definition area.
34518	FLG_ERR_MISSING_PROPERTY_LENGTH	Sequence number of property	The length value is required but missing for the indicated property in the definition area.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34519	FLG_ERR_MISSING_PROPERTY_PPNAME	Sequence number of property	The property short name is required but missing for the indicated property in the definition area.
34520	FLG_ERR_MISSING_REG_DPNAME	—	The DP NAME (DPNAME) property is required but missing in the input structure definition area.
34521	FLG_ERR_MISSING_REG_PTNAME	—	The PHYSICAL TYPE NAME (PTNAME) property is required but missing in the input structure definition area.
34522	FLG_ERR_MISSING_REG_CREATOR	—	The CREATOR property is required but missing in the input structure definition area.
34523	FLG_ERR_MISSING_REG_UPDATIME	—	The LAST CHANGED DATE AND TIME (UPDATIME) property is required but missing in the input structure definition area.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34524	FLG_ERR_MISSING_REG_UPDATEBY	—	The LAST CHANGED BY (UPDATEBY) property is required but missing in the input structure definition area.
34525	FLG_ERR_MISSING_REG_NAME	—	The EXTERNAL NAME OF OBJ TYPE (NAME) property is required but missing in the input structure definition area.
34526	FLG_ERR_MISSING_UUI_SEQUENCE	—	The indicated UUI sequence number was specified in the definition area, although the preceding number was not. UUI sequence numbers must not skip numbers in the sequence: 1, 2, and 3 is valid; 1, 3, and 5 is invalid.
34527	FLG_ERR_MISSING_RQD_INSTIDNT	—	The Instance identifier (INSTIDNT) property is required but missing in the input structure definition area.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34528	FLG_ERR_MISSING_RQD_NAME	—	The Name (NAME) property is required but missing in the input structure definition area.
34529	FLG_ERR_MISSING_RQD_OBJTYPID	—	The Object type identifier (OBJTYPID) property is required but missing in the input structure definition area.
34530	FLG_ERR_MISSING_RQD_UPDATEBY	—	The Last Changed By (UPDATEBY) property is required but missing in the input structure definition area.
34531	FLG_ERR_MISSING_RQD_UPDATIME	—	The Last Changed Date and Time (UPDATIME) property is required but missing in the input structure definition area.
34532	FLG_ERR_NOMATCH_PROPERTY_NAME	Sequence number of property	The indicated input property in the definition area matches the property short name for an existing property, but the property names do not match.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34533	FLG_ERR_NOMATCH_PROPERTY_SPEC	Sequence number of property	The indicated property in the definition area matches the property name and property short name for an existing property; however, the data type, length, value flag, or UUI sequence values do not match.
34534	FLG_ERR_PROPERTY_NOTEXIST	Sequence number of property	The property specified as part of the selection criteria does not exist.
34536	FLG_ERR_UNMATCH_DEFINITION	Sequence number of property	One of the following occurred: • The indicated property specified in the definition area for the object instance does not match any property defined for the object type. • The object instance has more properties defined in the definition area than are defined for the object type.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34537	FLG_ERR_PROPDUP	—	Duplicate property name or property short name specified in the definition area.
34538	FLG_ERR_REG_PROPS_OUT_OF_SEQUENCE	—	The registration properties are not specified in the correct sequence.
34539	FLG_ERR_RQD_PROPS_OUT_OF_SEQUENCE	—	The required properties are not specified in the correct sequence in the definition area.
34540	FLG_ERR_INV_V_FLAG_FOR_APPEND	Sequence number of property	The indicated appended property has a value flag of S or R. An appended property must have a value flag of ″O″ (optional property).
34541	FLG_ERR_INV_UUI_FOR_APPEND	Sequence number of property	The indicated appended property is specified as a UUI property. Appended properties cannot be UUI properties.
34542	FLG_ERR_NONBLANK_PROPERTY_V_FLAG	Sequence number of property	The value flag for the indicated property is not blank. The value flag is not used by this API call and must be left blank.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34543	FLG_ERR_NONBLANK_PROPERTY_CS_FLAG	Sequence number of property	The case-sensitivity flag for the indicated property is not blank. The case-sensitivity flag is not used by this API call and must be left blank.
34544	FLG_ERR_NONBLANK_PROPERTY_FS_FLAG	Sequence number of property	The fuzzy search flag for the indicated property is not blank. The fuzzy search flag is not used by this API call and must be left blank.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34545	FLG_ERR_NONBLANK_PROPERTY_UUISEQ	Sequence number of property	The UUI sequence position for the indicated property is not blank. The UUI sequence position is not used by this API and must be left blank. The data type is LONG VARCHAR and the UUI sequence position is not blank. A UUI property can be CHAR, VARCHAR, TIMESTAMP, but not LONG VARCHAR.
34546	FLG_ERR_NONBLANK_PROPERTY_RESERVED	Sequence number of property	The reserved area of the input structure property specifications must always be blank.
34547	FLG_ERR_UUI_V_FLAG_MUST_BE_R	Sequence number of property	The value flag for the indicated property is not valid because all UUI properties must have value flags of R (required).

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34548	FLG_ERR_AT_LEAST_ONE_UUI_PROP_RQD	—	None of the properties specified in the definition area are defined as UUI properties. Every Information Catalog Manager object type must be defined with at least one UUI property.
34550	FLG_ERR_DUP_REG_DPNAME	—	The DP NAME (DPNAME) specified in the definition area duplicates the DP NAME value of an existing object type registration. The DPNAME value must be unique across the Information Catalog Manager database.
34551	FLG_ERR_DUP_REG_PTNAME	—	The PHYSICAL TYPE NAME (PTNAME) duplicates the name of an existing table in the database. The PTNAME value must be unique across the Information Catalog Manager database.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34552	FLG_ERR_DUP_REG_NAME	—	The specified EXTERNAL NAME OF OBJ TYPE (NAME) duplicates the NAME value of an existing object type registration. The NAME must be unique across the Information Catalog Manager database.
34553	FLG_ERR_INV_DPNAME	—	The syntax of the specified DPNAME value is invalid.
34554	FLG_ERR_INV_DB_PTNAME	—	The specified PTNAME value is not valid according to database syntax rules.
34555	FLG_ERR_INV_DB_DPNAME	—	Reserved
34556	FLG_ERR_INV_DB_PROPERTY_PPNAME	—	The property short name is not valid according to database syntax rules.
34557	FLG_ERR_INV_TOT_PROPERTY_LEN	—	The total length of CHAR, VARCHAR, and TIMESTAMP properties, plus overhead, is longer than the maximum allowed by a database for each row in the physical table in the database.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34558	FLG_ERR_INV_PTNAME	—	The syntax of the specified PTNAME value is invalid.
34559	FLG_ERR_INV_PROPERTY_CS_FLAG_FOR_DB	Sequence number of property	The value for the case-sensitivity flag is not valid for the database.
34560	FLG_ERR_SRH_PROP_VAL_TOOLONG	Sequence number of property	The search criteria value is too long. The maximum length when using DB2 on OS/390 is 254 bytes.
34561	FLG_ERR_EXTRA_PROPS_IN_IOSTRUCT	—	The input structure contains one or more properties that are not in the object type definition.
34562	FLG_ERR_MISSING_REQ_PROPERTY	Sequence number of property	A required property is missing from the input structure of an FLGCreateInst or FLGUpdateInst API. The extended code points to the position of the missing property using the object type's complete definition.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34800	FLG_ERR_PROP_VALUE_REQUIRED	Sequence number of property	No value was specified in the object area for the indicated property. The definition for the property specifies that a value is required.
34801	FLG_ERR_PROP_VALUE_EXCEEDED	Sequence number of property	The length of the value for the indicated property exceeds the maximum length defined in the definition area.
34802	FLG_ERR_INVALID_PROPERTY_VALUE	Sequence number of property	The property value is invalid for one of the following reasons: • The value uses DBCS characters, but must use SBCS characters. • With FLGUpdateInst, the INSTIDNT value in the object area is not valid.
34803	FLG_ERR_INV_SRH_VAL_FOR_LONGVARCHAR	Sequence number of property	The search value for the indicated property is longer than the maximum length allowed for search criteria with a LONG VARCHAR data type (3000).

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34804	FLG_ERR_INV_OBJ_LENGTH	—	The actual length of the object area does not match the object length specified in the header area.
34805	FLG_ERR_PARMLIST_REQUIRES_HANDLES	Sequence number of property	The HANDLES property is not specified in the definition area.
34806	FLG_ERR_REG_CONFLICT	—	The DPNAME or the PTNAME values specified in the object area do not match the values for the registration information identified by the object type ID.
34807	FLG_ERR_ICON_EXCEEDS_LIMIT	—	The icon size is greater than the maximum icon size (30000).
34808	FLG_ERR_INST_VALUE_EXCEEDED	—	The total length of the instance value exceeds the database limit.
34809	FLG_ERR_INVALID_VARCHAR_LENGTH	—	Reserved

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
34810	FLG_ERR_INVALID_CREATOR	—	APIs FLGCreateInst and FLGUpdateInst found an error in the input I/O structure. The CREATOR value is not the same as the logged-on user ID. This is a requirement if the calling user is not authorized to perform object management operations.
35000	FLG_ERR_PRG_NOT_STARTED	—	The program could not be started due to an unexpected operating system error.
35001	FLG_ERR_PROG_PARM_TOOLONG	—	The parameter specified for the Parameter list (PARMLIST) property of the program object is too long for the platform-specific program invocation.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
35002	FLG_ERR_INV_PROG_PARM	—	The parameter list in the program object contains an unmatched token specifier (%), or a property delimited by token specifiers is not a property of the object type identified by the HANDLES property.
35003	FLG_ERR_PROGRAM_NOTEXIST	—	The program to be started does not exist or the path specification is incorrect.
35004	FLG_ERR_INV_SYNTAX_STARTCMD	—	The value of the STARTCMD property of the Program object is invalid.
36001	FLG_ERR_ACCESS_DENIED	—	Access is denied when opening or reading a file.
36002	FLG_ERR_BAD_INVOCATION	—	An error occurred on the Information Catalog Manager command line invocation.
36003	FLG_ERR_BROKEN_PIPE	—	Unable to open or read the specified file.
36004	FLG_ERR_BUFFER_OVERFLOW	—	Information Catalog Manager internal error.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
36005	FLG_ERR_CANNOT_MAKE	—	Unable to create the specified file.
36006	FLG_ERR_CLOSE_ERROR	—	Unable to close the file.
36007	FLG_ERR_COPY_ERROR	—	Unable to copy a file.
36008	FLG_ERR_DELETE_ERROR	—	Unable to delete the specified file.
36009	FLG_ERR_DEVICE_IN_USE	—	Unable to access a file; the file is currently in use.
36010	FLG_ERR_DIRECT_ACCESS_HANDLE	—	Information Catalog Manager internal error.
36011	FLG_ERR_DISK_FULL	—	The disk is full and the file cannot be created.
36012	FLG_ERR_DRIVE_LOCKED	—	Unable to access a drive; the drive is currently in use.
36013	FLG_ERR_DUPHNDL_ERROR	—	Information Catalog Manager internal error.
36014	FLG_ERR_EAS_DIDNT_FIT	—	The icon file has too many extended attributes.
36015	FLG_ERR_EA_LIST_INCONSISTENT	—	Some of the extended attributes of the icon file are invalid.
36016	FLG_ERR_EAS_NOT_SUPPORTED	—	Unable to copy a file with extended attributes to a file system that does not support extended attributes.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
36017	FLG_ERR_FILENAME_EXCED_RANGE	—	The file name or path was invalid.
36018	FLG_ERR_FILE_NOT_FOUND	—	The specified path and file name was not found.
36019	FLG_ERR_FINDFILE_ERROR	—	Unable to find the specified file.
36020	FLG_ERR_FINDNEXT_ERROR	—	Unable to find the next file.
36021	FLG_ERR_INVALID_ACCESS	—	Unable to write to the file; the file is read-only.
36022	FLG_ERR_INVALID_DIRECTORY	—	The specified directory is invalid.
36023	FLG_ERR_INVALID_DRIVE	—	Unable to access the specified drive.
36024	FLG_ERR_INVALID_EA_NAME	—	Information Catalog Manager internal error.
36025	FLG_ERR_INVALID_FILE_NAME	—	The specified file name is invalid.
36026	FLG_ERR_INVALID_FUNCTION	—	Information Catalog Manager internal error.
36027	FLG_ERR_INVALID_HANDLE	—	Information Catalog Manager internal error.
36028	FLG_ERR_INVALID_PARAMETER	—	Information Catalog Manager internal error.
36029	FLG_ERR_INVALID_TARGET_HANDLE	—	Information Catalog Manager internal error.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
36030	FLG_ERR_LOCK_VIOLATION	—	Unable to access a file; the file is locked by another application.
36031	FLG_ERR_META_EXPANSION_TOO_LONG	—	Information Catalog Manager internal error.
36032	FLG_ERR_MORE_DATA	—	Unable to open a file; the file is too large.
36033	FLG_ERR_NEED_EAS_FOUND	—	Unable to move the file to a drive that does not support extended attributes. Extended attributes are required for this file.
36034	FLG_ERR_NEGATIVE_SEEK	—	Information Catalog Manager internal error.
36035	FLG_ERR_NOT_DOS_DISK	—	The specified disk is not a valid disk or does not exist.
36036	FLG_ERR_NO_MORE_FILES	—	Information Catalog Manager internal error.
36037	FLG_ERR_NO_MORE_SEARCH_HANDLES	—	This Information Catalog Manager session reached the maximum number of handles. In your CONFIG.SYS file, increase the value for the FILES= option.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
36038	FLG_ERR_OPEN_ERROR	—	Unable to open the icon file, tag language file, echo file, or log file.
36039	FLG_ERR_OPEN_FAILED	—	Unable to open the icon file, tag language file, echo file, or log file.
36040	FLG_ERR_PATH_NOT_FOUND	—	The specified path was not found.
36041	FLG_ERR_PIPE_BUSY	—	Information Catalog Manager internal error.
36042	FLG_ERR_READ_ERROR	—	Information Catalog Manager internal error.
36043	FLG_ERR_SEEK_ON_DEVICE	—	Information Catalog Manager internal error.
36044	FLG_ERR_SETFILEPTR_ERROR	—	Information Catalog Manager internal error.
36045	FLG_ERR_SHARING_BUFFER_EXCEEDED	—	This file cannot be shared, because there is a buffer overflow.
36046	FLG_ERR_SHARING_VIOLATION	—	Unable to access this file. Another process is using this file.
36047	FLG_ERR_TOO_MANY_OPEN_FILES	—	Unable to open any more files. Under OS/2, increase the value of the FILES= option.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
36048	FLG_ERR_WRITE_ERROR	—	Information Catalog Manager internal error.
36049	FLG_ERR_WRITE_FAULT	—	Unable to write to the disk. The disk might be locked or unreadable.
36050	FLG_ERR_WRITE_PROTECT	—	Unable to write to the file. The file is read-only.
36200	FLG_ERR_NO_MORE_THREADS	—	No more system threads are available. Close some existing programs to continue.
36201	FLG_ERR_QDISK_FAIL	—	Unable to access information about the disk drive.
37001	FLG_ERR_INV_RESTART_OPT	—	The specified restart option (RestartOpt) was invalid. Valid values are B, C, b, or c.
37002	FLG_ERR_INV_OBJTYPE_OPT	—	The option on the ACTION.OBJTYPE tag is invalid. Valid options are MERGE, ADD, UPDATE, DELETE, DELETE_EXT, and APPEND.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37003	FLG_ERR_INV_OBJINST_OPT	—	The option on the ACTION.OBJINST tag is invalid. Valid options are ADD, UPDATE, DELETE, DELETE_TREE_REL, DELETE_TREE_ALL, and MERGE.
37004	FLG_ERR_INV_RELATION_OPT	—	The option on the ACTION.RELATION tag is invalid. Valid options are ADD and DELETE.
37005	FLG_ERR_TAG_OUT_OF_SEQUENCE	—	A tag is not in the correct sequence following an ACTION tag in the tag language file.
37006	FLG_ERR_KEYNAME_TOO_LONG	—	A UUI property short name on the INSTANCE tag is longer than the maximum length (8).
37007	FLG_ERR_INV_ACTION_TYPE	—	The keyword on the ACTION tag is invalid. Valid keywords are OBJTYPE, OBJINST, or RELATION.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37008	FLG_ERR_KEYWORD_TOO_LONG	—	A keyword on a tag is longer than the maximum allowed for the keyword.
37009	FLG_ERR_PROPNAME_TOO_LONG	—	Property short name on the INSTANCE tag is longer than the maximum length (8).
37010	FLG_ERR_VALUE_TOO_LONG	—	Value in the tag language file is longer than the maximum allowed by its keyword, property short name, or UUI property short name.
37011	FLG_ERR_OBJTAG_DUP_KEYWORD	—	A keyword on the OBJECT tag is specified more than once.
37012	FLG_ERR_PROPTAG_DUP_KEYWORD	—	A keyword on the PROPERTY tag is specified more than once.
37013	FLG_ERR_RELTAG_DUP_KEYWORD	—	A keyword is specified more than once on the RELTYPE tag.
37014	FLG_ERR_INSTTAG_DUP_KEYNAME	—	A UUI property short name is specified more than once on the INSTANCE tag.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37015	FLG_ERR_INSTTAG_DUP_PROPNAME	—	A property short name is specified more than once on the INSTANCE tag.
37016	FLG_ERR_OBJTAG_INV_KEYWORD	—	A keyword on the OBJECT tag is invalid. Valid keywords are TYPE, CATEGORY, EXTNAME, PHYNAME, ICOFILE and ICWFILE.
37017	FLG_ERR_PROPTAG_INV_KEYWORD	—	A keyword on the PROPERTY tag is invalid. Valid keywords are EXTNAME, DT, DL, SHRTNAME, NULLS, and UUISEQ.
37018	FLG_ERR_RELTAG_INV_KEYWORD	—	A keyword on the RELTYPE tag is invalid. Valid keywords are TYPE, SOURCETYPE, and TARGETYPE.
37019	FLG_ERR_CMMTTAG_INV_KEYWORD	—	A keyword on the COMMIT tag is invalid. The valid keyword is CHKPID.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37020	FLG_ERR_INSTTAG_INV_KEYNAME	—	A UUI property short name on the INSTANCE tag is invalid.
37021	FLG_ERR_INSTTAG_INV_PROPNAME	—	A property short name on the INSTANCE tag is invalid. The property short name must exist in the object type specified on the OBJECT tag.
37022	FLG_ERR_INSTTAG_MISSING_SKEY	—	SOURCEKEY is not the first keyword on the INSTANCE tag.
37023	FLG_ERR_INSTTAG_MISSING_TKEY	—	TARGETKEY is not the second keyword on the INSTANCE tag when creating or deleting a relationship.
37024	FLG_ERR_TAGFILE_PREMATURE_EOF	—	The Information Catalog Manager encountered the end of the tag language file unexpectedly when importing the tag language file.
37025	FLG_ERR_PROPTAG_INV_DT	—	The DT value on the PROPERTY tag is invalid. Valid values are C, V, L, and T.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37026	FLG_ERR_PROPTAG_RESERVED_SHRTNAME	—	The short name of a reserved property was specified as the value for SHRTNAME on the PROPERTY tag. The following short names are reserved and cannot be specified as the SHRTNAME: OBJTYPID, INSTIDNT, UPDATIME, and UPDATEBY.
37027	FLG_ERR_PROPTAG_INV_NULLS	—	NULLS value on the PROPERTY tag is invalid. Valid values are Y and N.
37028	FLG_ERR_PROPTAG_INV_UUISEQ	—	UUISEQ value on the PROPERTY tag is invalid. Valid values are 0, 1, 2, 3, 4, and 5.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37029	FLG_ERR_INSTTAG_RESERVED_PROPNAME	—	The property short name of a reserved property was specified on the INSTANCE tag. The following property short names are reserved and cannot be assigned values: OBJTYPID, INSTIDNT, UPDATIME, and UPDATEBY.
37030	FLG_ERR_OBJTAG_MISSING_REQD_KEYWORD	—	A required keyword is missing on the OBJECT tag.
37031	FLG_ERR_OBJTAG_KEYWORD_NOT_ALLOWED	—	A keyword specified on the OBJECT tag is not allowed with the current ACTION tag keyword and option.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37032	FLG_ERR_PROPTAG_MISSING_REQD_KEYWORD	—	A required keyword is missing on the PROPERTY tag. Required keywords are: EXTNAME, DT, DL, SHRTNAME, and NULLS. When NAME is specified as the value of SHRTNAME, SHRTNAME is the only required keyword.
37033	FLG_ERR_RELTAG_MISSING_REQD_KEYWORD	—	A required keyword is missing on the RELTYPE tag. Required keywords are TYPE, SOURCETYPE, and TARGETYPE.
37034	FLG_ERR_INVALID_DISKCNTL_TAG	—	The values and keywords on the DISKCNTL tag are invalid.
37035	FLG_ERR_NO_VALID_INPUT_TAG	—	The tag language file contains no valid tags.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37037	FLG_ERR_OBJTAG_INV_CATEGORY	—	The CATEGORY value on the OBJECT tag is invalid. Valid values are GROUPING, ELEMENTAL, CONTACT, DICTIONARY, and SUPPORT.
37038	FLG_ERR_RELTAG_INV_TYPE	—	The TYPE value on the RELTYPE tag is invalid. Valid values are CONTAIN, CONTACT, LINK, and ATTACHMENT.
37039	FLG_ERR_MISSING_LPAREN	—	A left parenthesis is missing following a keyword, UUI property short name, or property short name.
37040	FLG_ERR_INSTTAG_NO_PROPNAME	—	No property short names were specified on the INSTANCE tag.
37041	FLG_ERR_NO_VALUE	—	The value for the specified keyword is missing.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37042	FLG_ERR_NO_KEYWORD	—	A tag does not include any keywords. At least one keyword is required for all tags except COMMENT, NL, and TAB.
37043	FLG_ERR_TAG_FOLLOWED_BY_GARBAGE	—	A valid tag is followed by extra characters.
37044	FLG_ERR_BAD_PAREN_WITHIN_VALUE	—	A parenthesis specified within this value is invalid. A parenthesis within values must be surrounded by single quotation marks.
37046	FLG_ERR_PROPTAG_KEYWORD_NOT_ALLOWED	—	A specified keyword is not allowed on the PROPERTY tag when NAME is specified as the SHRTNAME value. Valid keywords in this case are SHRTNAME and UUISEQ.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37047	FLG_ERR_UNEXPECTED_LPAREN	—	A left parenthesis is specified before an expected keyword, UUI property short name, or property short name.
37048	FLG_ERR_UNEXPECTED_RPAREN	—	A right parenthesis is specified before an expected left parenthesis, keyword, UUI property short name, or property short name.
37300	FLG_ERR_CHKPT_DUP	—	Information Catalog Manager internal error.
37301	FLG_ERR_CHKPT_NOTEXIST	—	Information Catalog Manager internal error.
37302	FLG_ERR_INV_SAVEAREA_LEN	—	Information Catalog Manager internal error.
37303	FLG_ERR_INV_CHKPT_TOT_LEN	—	Information Catalog Manager internal error.
37304	FLG_ERR_MISSING_CHKPT_VALUE	—	Information Catalog Manager internal error.
37305	FLG_ERR_NO_MATCH_ON_CHKPTID	—	Unable to match the system-saved checkpoint ID with any COMMIT tag checkpoint ID in the specified tag language file.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37500	FLG_ERR_REQUEST_A_NEW_DISK_FAILED	—	The user did not insert the next tag language file diskette in the sequence.
37501	FLG_ERR_VERIFY_DISKETTE_SEQUENCE_FAILED	—	The Information Catalog Manager encountered an error while trying to verify the diskette sequence.
37502	FLG_ERR_UNABLE_TO_FIND_REQUIRED_PROPERTY	—	Unable to find a specified property short name in the target database. This property short name was specified on the INSTANCE tag while updating or merging an object instance using ACTION.OBJINST(UPDATE or ACTION.OBJINST(MERGE).
37503	FLG_ERR_UNABLE_TO_FIND_REQUIRED_OBJTYPE	—	Unable to find the object type name, specified on the OBJECT tag, in the target database.
37504	FLG_ERR_NONUNIQUE_UUI_KEY	—	The specified UUI values identify more than one instance.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37505	FLG_ERR_MISMATCH_UUI_IN_MERGE	—	In an object type merge, the UUI property short names for the object type in the input tag language file do not match the UUI property short names for the same object type in the Information Catalog Manager database.
37506	FLG_ERR_DATA_LENGTH_CONVERSION_FAILED	—	Information Catalog Manager internal error.
37507	FLG_ERR_MISMATCH_DATA_LENGTH_IN_MERGE	—	The value of DL (data length) on a PROPERTY tag following an ACTION.OBJTYPE(MERG tag in the input tag language file does not match the value for the same property in the target Information Catalog Manager database for the same object type.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37508	FLG_ERR_MISMATCH_DATA_TYPE_IN_MERGE	—	The value of DT (data type) on a PROPERTY tag following an ACTION.OBJTYPE(MERGE) tag in the input tag language file does not match the value for the same property in the target Information Catalog Manager database for the same object type.
37509	FLG_ERR_MISMATCH_PROPERTY_NAME_IN_MERGE	—	The value of SHRTNAME (property short name) on a PROPERTY tag that follows an ACTION.OBJTYPE(MERGE) tag in the input tag language filedoes not match any property in the Information Catalog Manager database for the same object type.
37510	FLG_ERR_MISMATCH_CATEGORY_IN_MERGE	—	The value of CATEGORY on an OBJECT tag following an ACTION.OBJTYPE(MERGE) tag in the input tag language file does not match the value in the Information Catalog Manager database for the same object type.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37511	FLG_ERR_MISSING_REQUIRED_OBJTYPE_MERGE_STATEMENT		Unable to merge an object instance using ACTION.OBJINST(MERGI before its object type is merged using ACTION.OBJTYPE(MERG The ACTION.OBJTYPE(MERG tag must be processed before an ACTION.OBJINST(MERGI for the same object type.
37512	FLG_ERR_NONUNIQUE_SOURCE_UUI_KEY	—	Reserved
37513	FLG_ERR_NONUNIQUE_TARGET_UUI_KEY	—	Reserved
37514	FLG_ERR_NO_TAGFILE_ON_DISKETTE	—	Unable to find the input tag language file on the provided diskette.
37515	FLG_ERR_WRONG_DISK_SEQUENCE	—	The diskettes containing the tag language file were inserted in the wrong order.
37516	FLG_ERR_REQ_INST_NOTFOUND	—	Unable to find the instance to be updated.
37801	FLG_ERR_NO_UUI	—	Export encountered an object with no UUI and cannot process.
37802	FLG_ERR_CREATEREG_FAILED	—	Reserved
37803	FLG_ERR_UPDATEREG_FAILED	—	Reserved

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37804	FLG_ERR_GETREG_FAILED	Reason code	Export calls FLGGetReg, which returned an error. See the log file for information about how this error affects the export.
37805	FLG_ERR_DELETEREG_FAILED	—	Reserved
37806	FLG_ERR_CREATETYPE_FAILED	—	Reserved
37807	FLG_ERR_APPENDTYPE_FAILED	—	Reserved
37808	FLG_ERR_GETTYPE_FAILED	—	Reserved
37809	FLG_ERR_DELETETYPE_FAILED	—	Reserved
37820	FLG_ERR_CREATEINST_FAILED	—	Reserved
37821	FLG_ERR_UPDATEINST_FAILED	—	Reserved
37822	FLG_ERR_GETINST_FAILED	Reason code	Export calls FLGGetInst, which returned an error. See the log file for information about how this error affects the export.
37823	FLG_ERR_DELETEINST_FAILED	—	Reserved
37824	FLG_ERR_LISTTYPE_FAILED	—	Reserved
37825	FLG_ERR_SEARCH_FAILED	—	Reserved
37826	FLG_ERR_RELATE_FAILED	—	Reserved
37827	FLG_ERR_LISTCONTACTS_FAILED	Reason code	Export calls FLGListContacts, which returned an error. See the log file for information about how this error affects the export.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37828	FLG_ERR_NAVIGATE_FAILED	Reason code	Export calls FLGNavigate, which returned an error. See the log file for information about how this error affects the export.
37829	FLG_ERR_FREEMEM_FAILED	Reason code	Export calls FLGFreeMem, which returned an error. See the log file for information about how this error affects the export.
37831	FLG_ERR_LISTASSOC_FAILED	Reason code	This function calls FLGListAssociates which returned an error.
37901	FLG_ERR_NULL_LOGFILE	—	The log file pointer parameter value is NULL. A value is required for this parameter.
37902	FLG_ERR_LOGFILE_OPENERR	Reason code	Import or export encountered an error while opening the log file. The extended code contains the reason code for the error.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37904	FLG_ERR_LOGFILE_WRITEERR	Reason code	Import or export encountered an error while writing to the log file. The extended code contains the reason code for the error.
37906	FLG_ERR_LOGFILE_CLOSEERR	Reason code	Import or export encountered an error while closing the log file. The extended code contains the reason code for the error.
37908	FLG_ERR_INV_TAGFILE_LEN	—	One of the following has occurred: • The specified name of the tag language file is null. • The full name of the tag language file including the path information, is longer than the maximum length allowed (259). • The tag language file name and extension are longer than the maximum length allowed (240).

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37909	FLG_ERR_INV_LOGFILE_LEN	—	One of the following has occurred: • The specified name of the log file is null. • The entire name, including the path, is longer than the allowed maximum length (259).
37910	FLG_ERR_INV_TAGFILE	—	The specified drive for the tag language file is invalid because the Information Catalog Manager encountered an error while trying to access it. If the tag language file is in MDIS format, then the drive cannot be a removable drive.
37911	FLG_ERR_INV_LOGFILE	—	The specified drive for the log file is invalid. The specified drive might be removable, or an error occurred when the Information Catalog Manager tried to access it.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37912	FLG_ERR_ECHOFILE_OPENERR	Reason code	Import encountered an error while opening the echo file. The extended code contains the reason code for the error.
37913	FLG_ERR_TAGFILE_READERR	Reason code	Import encountered an error while reading the tag language file. The extended code contains the reason code for the error.
37914	FLG_ERR_ECHOFILE_WRITEERR	Reason code	Import encountered an error while writing to the echo file. The extended code contains the reason code for the error.
37915	FLG_ERR_INV_ICOPATH_LEN	—	The specified icon path is too long. The maximum length for an icon path, including the drive and directories, is 246.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37919	FLG_ERR_ICOPATH_NONBLANK_EXT	—	The specified icon path (pszIcoPath) includes an extension. This value should include only the path.
37920	FLG_ERR_INV_ICOPATH	—	The drive or extension specified in the icon path is invalid for one of the following reasons: • The drive was not specified, the drive is removable, or the Information Catalog Manager encountered an error while reading from it. • A file extension was specified in the icon path.
37921	FLG_ERR_TAGFILE_OPENERR	Reason code	Import, export, or FLGXferTagBuf encountered an error while opening the tag language file. The extended code contains the reason code for the open error.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37922	FLG_ERR_TAGFILE_CLOSEERR	Reason code	Import, export, or FLGXferTagBuf encountered an error while closing the tag language file. The extended code contains the reason code for the error.
37923	FLG_ERR_ECHOFILE_CLOSEERR	Reason code	Import encountered an error while closing the echo file. The extended code contains the reason code for the error.
37924	FLG_ERR_INV_ECHOFILE_LEN	—	The length of the log file path with the tag language file name and the ECH extension is longer than the maximum length allowed for the complete echo file path and name. This maximum is 259 characters.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
37925	FLG_ERR_MAX_OBJTYPE_EXCEEDED	—	The tag language file contains more than the maximum number of discrete object types allowed (3500) when importing or exporting.
37926	FLG_ERR_TAGFILE_WRITEERR	Reason code	Export or the FLGXferTagBuf API encountered an error while trying to write to the tag language file. The extended code contains the reason code for the write error.
37928	FLG_ERR_INV_TAGFILE_EXT	—	The filename specified for the tag language file has an extension of ECH. This extension is invalid.
37929	FLG_ERR_INV_LOGFILE_EXT	—	The filename specified for the log file has an extension of ECH. This extension is invalid.
37930	FLG_ERR_TAGFILE_LOGFILE_CONFLICT	—	The specified log file is the same as the tag language file. The two files must be different.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
38000	FLG_ERR_INVALID_EXPORT_IOSTRUCT	Sequence number of object	The input structure for FLGExport is invalid.
38001	FLG_ERR_INVALID_CFLAG	Sequence number of object	The containee flag value is invalid in the FLGExport input structure. Valid values are Y or N.
38002	FLG_ERR_INVALID_TFLAG	Sequence number of object	The contact flag value is invalid in the FLGExport input structure. Valid values are Y or N.
38003	FLG_ERR_TAGFILE_EXIST	—	The name specified for the export output tag language file (pszTagFileID) points to a file that already exists. The name of the output tag language file must not already exist.
38004	FLG_ERR_GET_ICON_FAILED	Reason code	Unable to export the icon for the specified object type.
38005	FLG_ERR_INVALID_AFLAG	Sequence number of object	The attachment flag on the export input structure is not valid. Valid values are 'Y' or 'N'.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
38006	FLG_ERR_INVALID_LFLAG	Sequence number of object type.	The link flag in the export input structure is not valid. Valid values are 'Y' or 'N'.
39000	FLG_ERR_UPM_FAIL	—	The User Profile Management utility failed (logon failed or logon user ID is different than connected user ID).
39001	FLG_ERR_INV_INPUT_PARM	—	The input parameter keywords for the command are invalid or missing.
39002	FLG_ERR_MISSING_PARM_VALUE	—	The input parameter values for the command are invalid or missing.
39003	FLG_ERR_INIT_BIDI_ERROR	—	The Information Catalog Manager encountered an error while initializing for the bi-directional environment. This applies only when the Information Catalog Manager is running on an Arabic or Hebrew machine.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
39201	FLG_ERR_INVALID_USERTYPE_FOR_UPDATE	—	The user type specified to be updated is invalid. The valid types are either the primary or backup administrator.
39202	FLG_ERR_INVALID_USERTYPE_FOR_CRT_OR_DEL	—	The user type specified to be created or deleted is invalid. Only users authorized to perform object management tasks can be created or deleted.
39203	FLG_ERR_INVALID_ID_BAD_CHAR	—	The specified user ID contains an invalid character. Refer to your database documentation for valid characters.
39204	FLG_ERR_INVALID_ID_NUM_START	—	The specified user ID begins with a numeric. This is not a valid starting character.
39205	FLG_ERR_INVALID_ID_IMB_BLANK	—	The specified user ID contains an imbedded blank. This is not allowed.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
39206	FLG_ERR_INVALID_MUU_OPT	—	The option specified for the FLGManageUsers API is invalid. Valid actions are FLG_ACTION_CREATE, FLG_ACTION_UPDATE, FLG_ACTION_DELETE, or FLG_ACTION_LIST.
39209	FLG_ERR_INVALID_PADMIN_USERID	—	The specified user ID for the primary administrator is invalid. Verify the user ID syntax in your database documentation.
39210	FLG_ERR_INVALID_BADMIN_USERID	—	The specified user ID for the backup administrator is invalid. Verify the user ID syntax in your database documentation.
39211	FLG_ERR_INVALID_POWERUSER_USERID	Contains the index to the user ID in the input structure that is invalid.	The specified user ID is invalid. Verify the user ID syntax in your database documentation.
39502	FLG_ERR_CDF_ERROR	—	Reserved
39504	FLG_ERR_INSTPROFILE_ERROR	—	Reserved
39700	FLG_ERR_TERM_FAIL_ROLLBACK_CLOSE	—	Reserved
39701	FLG_ERR_TERM_FAIL_ROLLBACK	—	Reserved
39702	FLG_ERR_TERM_FAIL_COMMIT	—	Reserved

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
40001	FLG_ERR_INVALID_CONFIG_PROFILE	—	The MDIS Configuration profile file does not contain a valid BEGIN CONFIGURATION section.
40002	FLG_ERR_CONFIGFILE_READERR	Reason code	MDIS import encountered an error while reading the Configuration profile file.
40003	FLG_ERR_CONFIGFILE_CLOSEERR	Reason code	MDIS import encountered an error while closing the Configuration profile file.
40006	FLG_ERR_CONFIGFILE_INV_BEGIN_STMT	—	The MDIS Configuration profile file contains an invalid BEGIN statement. Valid statement is: BEGIN CONFIGURATION.
40007	FLG_ERR_CONFIGFILE_INV_END_STMT	—	The MDIS Configuration profile file contains an invalid END statement. Valid statement is: END CONFIGURATION.
40010	FLG_ERR_CONFIGFILE_INV_KEYWORD	—	The MDIS Configuration profile file contains an invalid keyword.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
40011	FLG_ERR_CONFIGFILE_INV_TEXT	—	The MDIS Configuration profile file contains invalid text.
40012	FLG_ERR_CONFIGFILE_INV_VALUE	—	The MDIS Configuration profile file contains an invalid keyword value.
40013	FLG_ERR_CONFIGFILE_VALUE_TOO_LONG	—	The MDIS Configuration profile file contains a keyword value that exceeds the maximum allowable length for that keyword.
40015	FLG_ERR_CONFIGFILE_PREMATURE_EOF	—	MDIS import unexpectedly encountered the end of the Configuration profile file.
40021	FLG_ERR_INVALID_TOOL_PROFILE	—	The MDIS Tool profile file does not contain a valid BEGIN TOOL section.
40022	FLG_ERR_TOOLFILE_READERR	Reason code	MDIS import encountered an error while reading the Tool profile file.
40023	FLG_ERR_TOOLFILE_CLOSEERR	Reason code	MDIS import encountered an error while closing the Tool profile file.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
40026	FLG_ERR_TOOLFILE_INV_BEGIN_STMT	—	The MDIS Tool profile file contains an invalid BEGIN statement. Valid statements are: BEGIN TOOL, BEGIN APPLICATIONDATA.
40027	FLG_ERR_TOOLFILE_INV_END_STMT	—	The MDIS Tool profile file contains an invalid END statement. Valid statements are: END TOOL, END APPLICATIONDATA.
40030	FLG_ERR_TOOLFILE_INV_KEYWORD	—	The MDIS Tool profile file contains an invalid keyword.
40031	FLG_ERR_TOOLFILE_INV_TEXT	—	The MDIS Tool profile file contains invalid text.
40032	FLG_ERR_TOOLFILE_INV_VALUE	—	The MDIS Tool profile file contains an invalid keyword value.
40033	FLG_ERR_TOOLFILE_VALUE_TOO_LONG	—	The MDIS Tool profile file contains a keyword value that exceeds the maximum allowable length for that keyword.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
40034	FLG_ERR_TOOLFILE_CONFLICTING_VALUES	—	The MDIS Tool profile file contains conflicting RECORD, DIMENSION, or ELEMENT values.
40050	FLG_ERR_TOOLFILE_PREMATURE_EOF	—	MDIS import unexpectedly encountered the end of the Tool profile file.
40100	FLG_ERR_UNSUPPORTED_MDIS_FUNCTION	—	The Configuration profile file specifies a function that is not supported by the Information Catalog Manager.
40101	FLG_ERR_MISSING_REQ_MDIS_KEYWORD	—	A required MDIS keyword is not present in the tag language file.
40110	FLG_ERR_TAGFILE_INV_KEYWORD	—	The MDIS tag language file contains an invalid keyword.
40111	FLG_ERR_TAGFILE_INV_TEXT	—	The MDIS tag language file contains invalid text.
40112	FLG_ERR_TAGFILE_INV_VALUE	—	The MDIS tag language file contains an invalid keyword value.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
40113	FLG_ERR_TAGFILE_VALUE_TOO_LONG	—	The MIDS tag language file contains a keyword value that exceeds the maximum allowable length for that keyword.
40115	FLG_ERR_MISSING_DQUOTE	—	A double quotation mark is missing following a keyword.
40116	FLG_ERR_UNEXPECTED_DQUOTE	—	A double quotation mark was found unexpectedly.
40117	FLG_ERR_SPECIFIED_PROPERTY_NOT_FOUND	—	Unable to find a specified property short name in the target database.
40118	FLG_ERR_TAGFILE_INV_END_STMT	—	The MDIS tag language file contains an invalid END statement.
40119	FLG_ERR_TAGFILE_INV_BEGIN_STMT	—	The MDIS tag language file contains an invalid BEGIN statement.
40130	FLG_ERR_INV_RECORD_SECTION	—	A BEGIN RECORD section is incorrectly nested in the MDIS tag language file.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
40131	FLG_ERR_INV_DIMENSION_SECTION	—	A BEGIN DIMENSION section is incorrectly nested in the MDIS tag language file.
40132	FLG_ERR_INV_SUBSCHEMA_SECTION	—	A BEGIN SUBSCHEMA section is incorrectly nested in the MDIS tag language file.
40201	FLG_ERR_DUPLICATE_IDENTIFIER	—	An identifier value is duplicated in the MDIS tag language file.
40202	FLG_ERR_INV_IDENTIFIER_REFERENCE	—	Either a SourceObjectIdentifier or a TargetObjectIdentifier value does not refer to an identifier value previously defined in the tag language file.
40211	FLG_ERR_INV_PART1_VALUE	—	The value for the first part of an MDIS object does not match the parent value.
40212	FLG_ERR_INV_PART2_VALUE	—	The value for the second part of an MDIS object does not match the parent value.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
40213	FLG_ERR_INV_PART3_VALUE	—	The value for the third part of an MDIS object does not match the parent value.
40214	FLG_ERR_INV_PART4_VALUE	—	The value for the fourth part of an MDIS object does not match the parent value.
40215	FLG_ERR_MDIS_WORK_BUFFER_OVERFLOW	—	An MDIS file (Configuration profile file, Tool profile file, or tag language file) contains a value that is longer than the maximum allowable size of internal work buffers (32700 bytes).
40216	FLG_ERR_MDIS_APPL_DATA_TOO_LONG	—	ApplicationData section of MDIS tag language file exceeds limits for Information Catalog Manager Application data object type. Information Catalog Manager Application data object type is limited to 10 properties of 32700 bytes each.
80000	FLG_SEVERR	—	Place holder; indicates the beginning of the numeric range for severe errors.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
80002	FLG_SEVERR_NO_MEMORY	—	The Information Catalog Manager is unable to allocate more memory.
80003	FLG_SEVERR_MEM_ERROR	—	One of the following occurred: • A hardware memory interrupt occurred. • Some corruption in the Information Catalog Manager heap prevents the Information Catalog Manager from allocating or deallocating memory.
80004	FLG_SEVERR_NO_CSA	—	Information Catalog Manager internal error.
80005	FLG_SEVERR_APIDLL_FAILURE	—	The API DLL is missing API calls, or the API DLL could not be loaded.
80006	FLG_SEVERR_VIOPOPUP_FAIL	—	The Information Catalog Manager is unable to display OS/2 character-based error messages using video input/output (VIO).

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
80007	FLG_SEVERR_BIDIDLL_FAILURE	—	The Information Catalog Manager encountered an error while loading the PMBIDI.DLL. This DLL is needed when the Information Catalog Manager runs on an Arabic or Hebrew machine.
80008	FLG_SEVERR_DG2IFORDLL_FAILURE	—	A necessary DG2IFOR.DLL file was not found or is invalid. The Information Catalog Manager cannot continue.
81000	FLG_SEVERR_STARTDBM_FAIL	—	Unable to start the local database management system. Refer to your database documentation for an explanation of the SQLCODE.
81001	FLG_SEVERR_STARTDB_FAIL	—	Reserved
81002	FLG_SEVERR_DB_DISCONNECTED	—	The database disconnected unexpectedly.
81003	FLG_SEVERR_DB_INCONSISTENT	—	The Information Catalog Manager detected an inconsistency in the Information Catalog Manager database.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
81004	FLG_SEVERR_COMMIT_FAIL	—	The commit call to the Information Catalog Manager database failed.
81005	FLG_SEVERR_ROLLBACK_FAIL	—	The rollback call to the Information Catalog Manager database failed.
81006	FLG_SEVERR_NO_DBSPACE	—	The database server has run out of space or the file system is full.
81007	FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE	Database SQLCODE	The Information Catalog Manager encountered a database error and rolled back any uncommitted changes to the database. Check the extended code for the database SQLCODE that describes the error condition that caused the Information Catalog Manager to perform the rollback.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
81008	FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL	Database SQLCODE	The Information Catalog Manager encountered a database error and attempted to roll back any uncommitted changes to the database, but this roll back failed. Check the extended code for the database SQLCODE that describes the error condition that caused the Information Catalog Manager to perform the rollback. The database might be in an inconsistent state and need to be recovered.
82000	FLG_SEVERR_INIT_FAIL	—	The Information Catalog Manager encountered an unexpected condition, probably an OS/2 internal memory error, that prevents the Information Catalog Manager from running normally.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
82001	FLG_SEVERR_TERM_FAIL	—	The Information Catalog Manager encountered an unexpected condition, probably an OS/2 internal memory error, that prevents the Information Catalog Managerfrom releasing its allocated resources. The resources will be freed when the calling application session ends.
82002	FLG_SEVERR_TERM_FAIL_CLOSE	—	Reserved
82200	FLG_SEVERR_GETREG_FAILED	Reason code	Export calls FLGGetReg, which returned a severe error.
82201	FLG_SEVERR_GETINST_FAILED	Reason code	Export calls FLGGetInst, which returned a severe error.
82202	FLG_SEVERR_LISTCONTACTS_FAILED	Reason code	Export calls FLGListContacts, which returned a severe error.
82203	FLG_SEVERR_NAVIGATE_FAILED	Reason code	Export calls FLGNavigate, which returned a severe error.
82204	FLG_SEVERR_FREEMEM_FAILED	Reason code	Export calls FLGFreeMem, which returned a severe error.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
82400	FLG_SEVERR_THREAD_FAILED	—	A severe error occurred while creating the new thread and Information Catalog Manager cannot continue.
82500	FLG_SEVERR_PARMS_MISSING	—	The Information Catalog Manager required system table is corrupted or missing.
82501	FLG_SEVERR_DGEMPTY	—	The Information Catalog Manager database contains no registrations or object types. The Information Catalog Manager database is corrupted. Recover the database using your backed-up database files.
82502	FLG_SEVERR_TYPE_WOUT_PROPERTY	—	No properties exist for the specified object type, or the Information Catalog Manager is unable to retrieve any properties.
82503	FLG_SEVERR_MORE_THAN_ONE_KA	—	A security violation occurred; more than one administrator is logged on at the same time.
83000	FLG_SEVERR_SESSION_ABENDED	—	Reserved

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
83001	FLG_SEVERR_CDF_ERROR	—	Reserved
83002	FLG_SEVERR_INTERNAL_ERROR	—	Reserved
84000	FLG_SEVERR_DEMO_EXPIRED	—	The evaluation period for IBM Information Catalog Manager Administrator has ended. Please contact the local software reseller or your IBM representative to order the product.
84101	FLG_SEVERR_DB_CONNECT_FAILED	—	Unable to connect to database. Refer to your database documentation for an explanation of the SQLCODE.
84102	FLG_SEVERR_DB_BIND	—	Unable to bind the Information Catalog Manager to the information catalog. The Information Catalog Manager has encountered an unexpected database error or cannot find the bind file in the current directory or path.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
84103	FLG_SEVERR_INSAUTH_BIND	—	You must have SYSADM authority to bind the Information Catalog Manager to the information catalog.
84104	FLG_SEVERR_CREATETAB	—	Unable to create the Information Catalog Manager system table.
84105	FLG_SEVERR_INSAUTH_GRANT	—	You must have SYSADM authority to grant access to the information catalog.
84106	FLG_SEVERR_CREATECOLLECTION	—	The Information Catalog Manager failed to create an AS/400® library collection.
84107	FLG_SEVERR_ICON_NOT_GENERATED	—	The Information Catalog Manager has encountered a system error, or is unable to find the Information Catalog Manager icon files or the Information Catalog Manager executable file. The Information Catalog Manager icons will not be generated.

Table 24. Information Catalog Manager reason codes (continued)

Number	Reason code	Extended codes	Explanation
84108	FLG_SEVERR_DGCOL_NOTEXIST	—	You must create the AS/400 library collection, Information Catalog Manager, prior to invoking this utility.
84109	FLG_SEVERR_DB_NOTFOUND	—	The Information Catalog Manager cannot find the specified database. Create the database if it does not exist. Then, register the remote database on your workstation.

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Trademarks

The following terms, which may be denoted by an asterisk(*), are trademarks of International Business Machines Corporation in the United States, other countries, or both.

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, other countries, or both.

UNIX is a registered trademark in the United States, other countries or both and is licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk(**) may be trademarks or service marks of others.

Glossary

A

administrator. A person responsible for managing the content and use of the Information Catalog Manager.

anchor. A Grouping object that contains other objects, but is not contained by another Grouping object.

Attachment. The category for object types used to attach additional information to another Information Catalog Manager object. For example, you can attach comments to an object.

B

browse. To display information catalog objects that are grouped by subject. Contrast with *search*.

C

catalog. See *information catalog* and *database catalog*.

category. A classification for Information Catalog Manager object types. The category designates the:

- Actions available to object types
- Relationships allowed between object types in the same or different categories.

Object types belong to one of the following categories:

Attachment
Contact
Dictionary
Elemental
Grouping
Program
Support

CelDial sample data. A sample information catalog (ICMSAMP) available when you install the Information Catalog Manager that can be used for installation verification. This sample information catalog is also used in the exercises in the*Information Catalog Manager User's Guide*.

collection. A container for objects. A collection can be used to gather objects of interest for easy access.

Comments. A classification for objects that annotate another object in the Information Catalog Manager. For example, you may want to attach a Comments object to a chart object that contains notes about the data in the chart.

The Comments object type is with the Information Catalog Manager. You cannot add properties to it.

commit. To make changes to information catalog database permanent. Contrast with *roll back*.

contact. A reference for more information about an object. Further information might include the person who created the information that the object represents, or the department responsible for maintaining the information.

Contact. A category for the Contact object type and other object types that identify contacts.

Contact object type. A classification for objects that identify contacts.

D

database catalog. A collection of tables that contains descriptions of database objects such as tables, views, and indexes.

DBCS. Double-byte character set.

decision-support system. A system of applications that help users make decisions. This kind of system allows users to work with information presented in meaningful ways; for example, spreadsheets, charts, and reports.

delete history. A log of delete activity, the capture of which is turned on and off by the Information Catalog Manager administrator. The log can be transferred to a tag language file.

derived data. Data that is copied or enhanced (perhaps by summarizing the data) from operational data sources into an informational database.

descriptive data. Data that identifies and describes an object, for example, the name of a table, the location of a spreadsheet, or the creator of a document. Also called metadata.

Description view. A view that lists the properties and property values for an object.

Dictionary. The category for object types that can be used to define terminology (for example, the "Glossary entries" object type in the sample information catalog).

dictionary facility. A collection of definitions or synonyms for the business terms you use in the information catalog. After it is created, the dictionary facility appears in every user's Catalog window as a saved search icon.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Contrast with *single-byte character set*.

DP NAME. An identification for an object type that uniquely identifies it for import operations. Also called the short name of an object type.

E

echo file. A file produced by the Information Catalog Manager when it imports a tag language file. This file contains all the tags that have been processed since either the beginning of the tag language file or the point when the last COMMIT tag was processed.

Elemental. The category for non-Grouping object types that are the building blocks for other Information Catalog Manager object types. Elemental object types are at the bottom of object type hierarchies. "Columns in relational tables," "Presentations {electronic and hardcopy}," and "Graphics and Images" are all examples of Elemental object types.

export. To copy metadata from the Information Catalog Manager, translate the metadata into tag language, and put this output in a tag language file for a subsequent import operation.

external name. The 80-byte name for an object type. Also called object type name.

extract control file. A file that contains statements that control the operation of an extractor utility program.

extract program. A utility program that copies from a metadata source, such as an RDBMS catalog, translates the metadata into tag language, and places this output in a tag language file.

F

FAT. File allocation table. A table used to allocate space on a disk for a file and to locate the file.

FLGID. See *object identifier.*

G

Grouping. The category for object types that can contain other object types. Examples of Grouping object types available in the sample information catalog included with the Information Catalog Manager are: "Tables or views in a relational database," which contains the Elemental object type "Columns in relational tables"; and "Multi-dimensional model," which contains another Grouping object type "Dimension."

H

HPFS. High-performance file system. In OS/2, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. File names used with the HPFS can have as many as 254 characters.

I

import. To apply the contents of a tag language file to an Information Catalog Manager to initially populate the information catalog, change the information catalog contents, or copy the contents of another information catalog to the information catalog.

information catalog. The database managed by the Information Catalog Manager containing descriptive data that helps users identify and locate the data and information available to them in the organization.

Information Catalog Manager application program interface (API). The portion of the Information Catalog Manager that processes application program requests for the Information Catalog Manager services and functions.

information source. An item of data or information, such as a table or chart, that is represented by an Information Catalog Manager object.

informational application. A program or system that lets users retrieve and analyze their data.

informational database. A database that contains derived data and is intended for business decision making.

input structure. A self-defining data structure used to submit data to the Information Catalog Manager application program interface.

instance. See *object*.

instance identifier. A 10-digit numeric identifier generated by the Information Catalog Manager for each object. The identifier is unique for that object within a given object type (an object of another object type may have the same identifier), and within a given information catalog database (an object in another information catalog database may have the same identifier).

I/O structure. See *input structure* and *output structure*.

K

keyword. An element of the Information Catalog Manager tag language that identifies the meaning of a data value imported into or exported out of an information catalog.

keyword search. See *search*.

L

link. A connection between two or more objects involved in a linked relationship.

linked relationship. A relationship between objects in an information catalog. Objects in a linked relationship are peers, rather than one an underlying object of the other.

For example, in the sample information catalog included with the Information Catalog Manager, the object called **CelDial Sales Information** is linked with various objects describing CelDial advertisements for the year.

log file. A file produced by the Information Catalog Manager when it imports a tag language file or exports objects in the information catalog. This file records the times and dates when the import or export started and stopped and any error information for the process.

M

metadata. Data about information sources. See *descriptive data*.

multiple character wildcard. A character used to represent any series of characters of any

length. By default, the multiple character wildcard is an asterisk (*). See also *wildcard* and *single character wildcard*.

N

not-applicable symbol. A character that indicates that a value for a required property was not provided when an object was created. The not-applicable symbol is a hyphen (-) by default, but you could have identified a different symbol when you created the information catalog.

O

object. An item that represents a unit or distinct grouping of information. Each Information Catalog Manager object identifies and describes information, but does not contain the actual information. For example, an object can provide the name of a report, list its creation date, and describe its purpose.

object identifier. A 16-digit identifier for an object that is made up of its 6-digit object type identifier and its 10-digit instance identifier that is used with some API calls. See *object type identifier* and *instance identifier*.

object type. A classification for objects. An object type is used to reflect a type of business information, such as a table, report, or image.

The Information Catalog Manager provides a set of sample object types, which you can modify. You can also create additional object types to meet the needs of your organization.

object type identifier. A 6-digit numeric identifier generated by the Information Catalog Manager for each object type. The identifier is unique within the information catalog database.

object type registration. With the Information Catalog Manager application program interface, the basic information about an object type that you must define in the Information Catalog Manager before you can define the properties for the object type. This information includes the category, the name, the icon, and the name of the table containing the object information.

operational data. Data used to run the day-to-day operations of an organization.

option. In Information Catalog Manager tag language, a parameter of the ACTION tag that defines the action to be performed on objects or object types in the î database when the tag language file is imported.

output structure. A self-defining data structure produced by the Information Catalog Manager when returning data produced by an Information Catalog Manager API call.

P

physical type name. The name of the table in the information catalog database that contains metadata for instances of a specific object type.

populate. To add object types, objects, or metadata to the Information Catalog Manager.

Program category. The category for the Programs object type.

Programs object type. A classification for objects that identify and describe applications capable of processing the actual information described by Information Catalog Manager objects.

The Programs object type is included with the Information Catalog Manager.

property. A characteristic or attribute that describes a unit of information. Each object type has a set of associated properties. For example, the "Graphics and Images" object type in the sample information catalog includes the following properties:

Name
Description
Image type
Image filename

For each object, a set of values are assigned to the properties.

property name. The 80-byte descriptive name of a property that is displayed in the Information Catalog Manager user interface. Contrast with *property short name*.

property short name. An 8-character name used by the Information Catalog Manager to uniquely identify a property of an object or object type.

property value. The value of a property.

PT NAME. See *physical type name.*

R

RDBMS. Relational database management system.

RDBMS catalog. A set of tables that contain descriptions of SQL objects, such as tables, views, and indexes, maintained by an RDBMS.

relational database management system. A software system, such as DB2 UDB for OS/2, that manages and stores relational data.

registration. See *object type registration*.

roll back. To remove uncommitted changes to the information catalog database. Contrast with *commit*.

S

saved search. A set of search criteria that is saved for subsequent use. Appears as an icon in the Catalog window.

SBCS. Single-byte character set.

search. To request the display of the Information Catalog Manager objects that meet specific criteria.

search by subject. See *browse.*

search by term. See *search.*

search criteria. Options and character strings used to specify how to perform a search. This can include object type names, property values, whether the search is for an exact match, and whether the search is case sensitive.

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set*.

single character wildcard. A character used to represent any single character. By default, the single character wildcard is a question mark (?). See also *wildcard* and *multiple character wildcard*.

subject search. See *browse.*

Support. The category for object types that provide additional information about your information catalog or enterprise (for example, the " Information Catalog Manager News" object type in the sample information catalog).

support facility. A collection of information you consider helpful for users of your information catalog, such as announcements of changes or updates to the information catalog. After it is created, the support facility appears in every user's Catalog window as a saved search icon.

T

tag. An element of the tag language. Tags indicate actions to be taken when the tag language file is imported to the information catalog.

tag language. A format for defining object types and objects, and actions to be taken on those object types and objects, in the Data Warehouse Center or the information catalog.

tag language file. A file that contains tag language that describes objects and object types to be added, updated or deleted in the Data Warehouse Center or in the information catalog, when the file is imported. A tag language file is produced by exporting objects from the Data Warehouse Center or from the Information Catalog Manager.

In the Information Catalog Manager, a tag language file is also produced by:
- Transferring a delete history log.

- Extracting descriptive data from another database system using an extract program.

Tree view. A view that displays hierarchically an object and the objects it contains.

U

unit of work. A recoverable sequence of operations within an application process. A unit of work is the basic building block a database management system uses to ensure that a database is in a consistent state. A unit of work is ended when changes to the database are committed or rolled back.

universal unique identifier (UUI). A key for an object. The key is comprised of up to five properties, which, when concatenated in a designated order, uniquely identify the object during import and export functions.

user. A person who accesses the information available in the information catalog but who is not an administrator.

Some Information Catalog Manager users, if they have been granted authority, can perform some object management tasks normally performed by administrators.

W

wildcard. A special character that is used as a variable when specifying property values in a search. See also *single character wildcard* and *multiple character wildcard*.

Bibliography

To get copies of the books listed here, or to get more information about a particular library, see your IBM representative.

Data Warehouse Center publications

Warehouse Manager Installation Guide (GC26–9998)

Information Catalog Manager Administration Guide (SC26–9995)

Data Warehouse Center Administration Guide (SC26–9993)

Information Catalog Manager User's Guide (SC26–9996)

IBM DB2 Universal Database Message Reference (GC09-2978)

Bibliography

Index

Special Characters

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. This guide suggests information that you can gather to help DB2 Customer Support to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

ftp.software.ibm.com
Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l
These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2
Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the *IBM Software Support Handbook*. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Program Number: 5648-D35

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9997-00

Spine information:

IBM® DB2® Warehouse Manager

Information Catalog Manager Programming Guide and Reference

Version 7