

IBM DB2 Information Integrator



Wrapper Developer's API Reference

Version 8

IBM DB2 Information Integrator



Wrapper Developer's API Reference

Version 8

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 237.

This document contains proprietary information of IBM. It is provided under a license agreement and Copyright law protects it. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative:

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

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

© **Copyright International Business Machines Corporation 2003. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book.	v	Wrapper_Utilities class	214
Who should read this book	v		
Conventions and terminology used in this book	v		
C++ classes and methods	1	DB2 Information Integrator technical documentation	223
Catalog classes for the C++ API.	1	Accessing books and release information	223
Catalog_Option class	2	DB2 Information Integrator books	223
Wrapper_Info class	4	Release notes and installation requirements	226
Server_Info class	13	Fix packs for DB2 Information Integrator documentation.	226
User_Info class	24	Accessing topics using the DB2 Information Integrator Information Center or the DB2 HTML Documentation CD.	227
Column_Info class	30	Features of the DB2 Information Integrator Information Center.	227
Nickname_Info class	52	Finding topics in the DB2 Information Integrator Information Center.	227
Wrapper classes for the C++ API	67	Using the DB2 HTML documentation	229
Unfenced_Generic_Wrapper class	67	Searching the DB2 documentation	231
Fenced_Generic_Wrapper class.	75	Troubleshooting DB2 documentation search with Netscape 4.x	232
Server classes for the C++ API	81		
Unfenced_Generic_Server class.	82	Accessibility	235
Fenced_Generic_Server class	93	Keyboard input and navigation	235
User classes for the C++ API	100	Accessible display	235
Unfenced_Generic_User class	100	Font settings	235
Fenced_Generic_User class.	106	Nondependence on color	235
Nickname classes for the C++ API	109	Alternative alert cues	235
Unfenced_Generic_Nickname class	110	Compatibility with assistive technologies	236
Fenced_Generic_Nickname class	119	Accessible documentation	236
Remote_Connection class	124		
Operation classes for the C++ API	131	Notices	237
Remote_Query class	132	Trademarks	239
Remote_Passthru class	142		
Request classes for the C++ API	149	Index	241
Request class	150		
Reply class	156	Contacting IBM	243
Request_Exp class.	173	Product information	243
Request_Exp_Type class.	180	Comments on the documentation	243
Request_Constant class	184		
Predicate_List class	188		
Data classes for the C++ API	195		
Runtime_Data_Desc class	195		
Runtime_Data_Desc_List class	200		
Runtime_Data class	203		
Runtime_Data_List class	212		

About this book

This book provides reference information about API classes that you can use when you develop a wrapper for a data source. An overview of each class is presented with general descriptions and usage information. The member functions (and constructors and destructors, if applicable) of each class are then listed with the corresponding purpose, syntax, return values, and necessary input and output arguments. After you develop a wrapper for a data source, you can use the data source in a federated database system.

Who should read this book

This book is intended for DBAs and wrapper developers who use APIs with DB2® Information Integrator that is offered by IBM®.

Conventions and terminology used in this book

Highlighting Conventions:

This book uses these highlighting conventions:

Boldface type

Indicates commands and graphical user interface controls (such as names of fields, names of push buttons, and menu choices). Boldface type is used to designate notes, restrictions, prerequisites, and recommendations.

`Monospace type`

Indicates text that you type, file names, and code examples. Monospace type is also used for SQL statement or DB2 command parameter names.

Italic type

Indicates SQL statement or DB2 command parameter values that you replace with an appropriate value. SQL statement or DB2 command examples use italic type for sample parameter values. Italic type is used to emphasize words, to identify new terms, and to indicate document titles.

UPPERCASE TYPE

Indicates the names of DB2 commands and SQL statements, and their keywords. Uppercase is also used for data type names, options, and acronyms.

C++ classes and methods

The following sections describe the classes that you can use with the C++ API. These classes include:

- Catalog classes
- Wrapper classes
- Server classes
- User classes
- Nickname classes
- Remote connection classes
- Operation classes
- Request classes
- Data classes
- Wrapper utilities classes

An overview of each class is presented with general descriptions and usage information. The member functions (and constructors and destructors, if applicable) of each class are then listed with the corresponding purpose, syntax, return values, and necessary input and output arguments.

Catalog classes for the C++ API

The following table describes each catalog class for the C++ API.

Table 1. Catalog classes

Class name	Description
Catalog_Option	The class that represents a named catalog option, either single-valued or multi-valued. Subclasses distinguish between these. Do not instantiate or subclass this class.
Wrapper_Info	A subclass of Catalog_Info that contains catalog information for a wrapper.
Server_Info	The class that encapsulates the catalog information for a server object (data from the CREATE SERVER and ALTER SERVER statements).
User_Info	The class that encapsulates the catalog information for a user mapping from the CREATE USER MAPPING and ALTER USER MAPPING statements.

Table 1. Catalog classes (continued)

Class name	Description
Column_Info	The class that encapsulates catalog information for a column of a nickname. This class includes column statistical information.
Nickname_Info	The class that encapsulates a nickname definition from the catalog and includes column and index definitions.

Related reference:

- “Catalog_Option class” on page 2
- “Wrapper_Info class” on page 4
- “Server_Info class” on page 13
- “User_Info class” on page 24
- “Column_Info class” on page 30
- “Nickname_Info class” on page 52

Catalog_Option class

This topic describes the Catalog_Option class and provides details for each member function.

Overview

The Catalog_Option class represents a named catalog option, either single-valued or multi-valued. Subclasses distinguish between these.

The Catalog_Option class is one of the catalog classes for the C++ API.

Usage The DB2 federated server instantiates the named catalog options for each option that is specified in the catalog or on a Data Definition Language (DDL) statement. The wrapper can instantiate these named catalog options (through the add_option() method of a Catalog_Info subclass) to add or change option values.

File sqlqg_catalog.h

Data members
None.

Member functions

The following table describes each member function of the Catalog_Option class. Each function is described in more detail after the table.

Table 2. Member functions for the Catalog_Option class

Member function	Description
get_name	Retrieve the option name.
get_action	Retrieve the action (ADD, SET, DROP, or none) for this option.
get_value	Retrieve the option value.
get_value	Retrieve the option value and length.

get_name function**Purpose**

Retrieve the option name.

Syntax

```
sqluint8* get_name ()
```

Input arguments

None.

Output arguments

None.

Return value

Null-terminated option name.

get_action function**Purpose**

Retrieve the action (ADD, SET, DROP, or none) for this option.

Syntax

```
Catalog_Option::Action get_action ()
```

Input arguments

None.

Output arguments

None.

Return value

Action.

get_value function**Purpose**

Retrieve the option value.

Catalog_Option

Syntax

```
virtual sqluint8* get_value ()
```

Input arguments

None.

Output arguments

None.

Return value

Null-terminated option value.

get_value function

Purpose

Retrieve the option value and length.

Syntax

```
virtual sqluint8* get_value (sqlint32* a_length)
```

Input arguments

None.

Output arguments

Table 3. Output arguments for the get_value member function

Name	Data type	Description
a_length	sqlint32*	Length of the option value.

Return value

Null-terminated option value.

Related reference:

- “Catalog classes for the C++ API” on page 1

Wrapper_Info class

This topic describes the Wrapper_Info class and provides details for the constructor and the member functions.

Overview

The Wrapper_Info class is a subclass of Catalog_Info and contains catalog information for a wrapper.

The Wrapper_Info class is one of the catalog classes for the C++ API.

Usage This class is instantiated by the DB2 federated server to contain

information from a CREATE WRAPPER statement or from the DB2 Information Integrator catalog. This class is instantiated by the wrapper when information is added during CREATE WRAPPER or ALTER WRAPPER operations.

File sqlqg_catalog.h

Data members

None.

Constructors and member functions

The following tables describe the constructor and the member functions of the Wrapper_Info class. The constructor and functions are described in more detail after the tables.

Table 4. Constructors for the Wrapper_Info class

Constructor	Description
Wrapper_Info	Construct an instance of Wrapper_Info.

Table 5. Member functions for the Wrapper_Info class

Member function	Description
get_wrapper_name	Retrieve the wrapper name.
get_corelib	Retrieve the name of the wrapper library.
set_type	Set the wrapper type.
get_type	Retrieve the wrapper type.
set_version	Set the wrapper code version.
get_version	Retrieve the wrapper version, if present.
copy	Duplicate a Wrapper_Info object.
add_option	Add an option to the catalog information.
drop_option	Delete an option from a Catalog_Info class.
get_option	Retrieve a catalog option by name.
get_first_option	Retrieve a pointer to the first option in the option chain.
get_next_option	Retrieve the next option in the option chain.

Wrapper_Info constructor

Purpose

Construct an instance of Wrapper_Info.

Syntax

Wrapper_Info

Wrapper_Info ()

Input arguments

None.

Output arguments

None.

get_wrapper_name function

Purpose

Retrieve the wrapper name.

Syntax

```
sqlint32 get_wrapper_name (sqluint8** a_name)
```

Input arguments

None.

Output arguments

Table 6. Output arguments for the get_wrapper_name member function

Name	Data type	Description
a_name	sqluint8**	Pointer to the null-terminated wrapper name string.

Return value

Return code. The value is 0 if the name is present. The value is SQLQG_NOVALUE if the name is not present.

get_corelib function

Purpose

Retrieve the name of the wrapper library. This name is the base name of the wrapper library from the CREATE WRAPPER statement.

Syntax

```
sqlint32 get_corelib (sqluint8** a_lib_name)
```

Input arguments

None.

Output arguments*Table 7. Output arguments for the get_corelib member function*

Name	Data type	Description
a_lib_name	sqluint8**	Pointer to the null-terminated library name string.

Return value

Return code. The value is 0 if the name is present. The value is SQLQG_NOVALUE if the name is not present.

set_type function**Purpose**

Set the wrapper type.

Syntax

```
void set_type (sqluint8 a_wrapper_type)
```

Input arguments*Table 8. Input arguments for the set_type member function*

Name	Data type	Description
a_wrapper_type	sqluint8	Wrapper type (must be N).

Output arguments

None.

Return value

None.

get_type function**Purpose**

Retrieve the wrapper type.

Syntax

```
sqlint32 get_type (sqluint8* a_wrapper_type)
```

Input arguments

None.

Wrapper_Info

Output arguments

Table 9. Output arguments for the `get_type` member function

Name	Data type	Description
<code>a_wrapper_type</code>	<code>sqluint8*</code>	Wrapper type.

Return value

Return code. The value is 0 if the type is present. The value is `SQLQG_NOVALUE` if the type is not present.

set_version function

Purpose

Set the wrapper code version.

Syntax

```
void set_version (sqlint32 a_wrapper_version)
```

Input arguments

Table 10. Input arguments for the `set_version` member function

Name	Data type	Description
<code>a_wrapper_version</code>	<code>sqlint32</code>	Wrapper code version.

Output arguments

None.

Return value

None.

get_version function

Purpose

Retrieve the wrapper version, if present.

Syntax

```
sqlint32 get_version (sqlint32* a_wrapper_version)
```

Input arguments

None.

Output arguments*Table 11. Output arguments for the get_version member function*

Name	Data type	Description
a_wrapper_version	sqlint32*	Wrapper code version.

Return value

Return code. The value is 0 if the version is present. The value is SQLQG_NOVALUE if the version is not present.

copy function**Purpose**

Duplicate a Wrapper_Info object.

Syntax

```
sqlint32 copy (Wrapper_Info** a_new_wrapper_info)
```

Input arguments

None.

Output arguments*Table 12. Output arguments for the copy member function*

Name	Data type	Description
a_new_wrapper_info	Wrapper_Info**	Pointer to the newly allocated Wrapper_Info.

Return value

Return code. A value of 0 indicates success.

add_option function**Where defined**

Catalog_Info

Purpose

Add an option to the catalog information.

Usage The wrapper can invoke this member function when wrapper-generated options are added to the catalog during CREATE WRAPPER and ALTER WRAPPER statement processing.

Syntax

```
sqlint32 add_option (sqluint8* a_opt_name,  
                    sqlint32 a_name_len,  
                    sqluint8* a_opt_value,  
                    sqlint32 a_value_len,  
                    Catalog_Option::Action a_action  
                    = Catalog_Option::sqlqg_None,  
                    char* a_option_type = "")
```

Input arguments

Table 13. Input arguments for the add_option member function

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_name_len	sqlint32	Length of the option name.
a_opt_value	sqluint8*	Option value (not null-terminated).
a_value_len	sqlint32	Length of the option value.
a_action	Catalog_Option::Action	Action for this option (ADD, SET, DROP, or none).
a_option_type	char*	Token that is used in the SQLN1884 error message if this is a duplicate option. For wrapper options, use the SQLQG_WRAPPER_OPTION constant that is defined in the sqlqg_misc.h header file.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

drop_option function

Where defined

Catalog_Info

Purpose

Delete an option from a Catalog_Info class. This member function does not drop the option from the catalog. The option is dropped from the catalog when an option to a delta Catalog_Info object is added with an action of Catalog_Option::sqlqg_Drop.

Syntax

```
sqlint32 drop_option (Catalog_Option* a_option)
```

Input arguments*Table 14. Input arguments for the drop_option member function*

Name	Data type	Description
a_option	Catalog_Option*	Option to be dropped.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_option function**Where defined**

Catalog_Info

Purpose

Retrieve a catalog option by name.

Usage The output argument option is null if the option is not found.**Syntax**

```
sqlint32 get_option (sqluint8*      a_opt_name,
                    sqlint32      a_name_len,
                    Catalog_Option** a_option)
```

Input arguments*Table 15. Input arguments for the get_option member function*

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_name_len	sqlint32	Length of the option name.

Output arguments*Table 16. Output arguments for the get_option member function*

Name	Data type	Description
a_option	Catalog_Option**	Found option.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the option was not found.

get_first_option function

Where defined

Catalog_Info

Purpose

Retrieve a pointer to the first option in the option chain.

Syntax

```
Catalog_Option* get_first_option ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the first option in the option chain. The value is null if the chain is empty.

get_next_option function

Where defined

Catalog_Info

Purpose

Retrieve the next option in the option chain.

Syntax

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

Input arguments

Table 17. Input arguments for the get_next_option member function

Name	Data type	Description
a_current_option	Catalog_Option*	Current option.

Output arguments

None.

Return value

Pointer to next option in the option chain. The value is null if at the end.

Related reference:

- “Catalog classes for the C++ API” on page 1

Server_Info class

This topic describes the `Server_Info` class and provides details for the constructors and the member functions.

Overview

The `Server_Info` class encapsulates the catalog information for a server object (data from `CREATE SERVER` and `ALTER SERVER` statements).

The `Server_Info` class is one of the catalog classes for the C++ API.

Usage This class is instantiated by the DB2 federated server to contain information from a `CREATE SERVER` or an `ALTER SERVER` statement or to contain information from the DB2 Information Integrator catalog. This class is also instantiated by the wrapper when information is added during `CREATE SERVER` or `ALTER SERVER` statement operations.

File `sqlqg_catalog.h`

Data members

None.

Constructors and member functions

The following tables describe the constructors and the member functions of the `Server_Info` class. The constructors and functions are described in more detail after the tables.

Table 18. Constructors for the `Server_Info` class

Constructor	Description
<code>Server_Info</code>	Construct an empty <code>Server_Info</code> object.
<code>Server_Info</code>	Construct a <code>Server_Info</code> object with specified parameters.

Table 19. Member functions for the `Server_Info` class

Member function	Description
<code>add_option</code>	Add a single-value option to the catalog information.
<code>drop_option</code>	Delete an option from a <code>Catalog_Info</code> class.
<code>get_option</code>	Retrieve a catalog option by name.
<code>get_first_option</code>	Retrieve a pointer to the first option in the option chain.
<code>get_next_option</code>	Retrieve the next option in the option chain.

Server_Info

Table 19. Member functions for the `Server_Info` class (continued)

Member function	Description
<code>get_basic_info</code>	Retrieve basic information from a <code>Server_Info</code> object.
<code>get_server_name</code>	Retrieve the server name from <code>Server_Info</code> , if the name is valid.
<code>set_server_type</code>	Set the server type for the server.
<code>get_server_type</code>	Retrieve the server type from <code>Server_Info</code> , if the type is valid.
<code>set_server_version</code>	Set the server version for the server.
<code>get_server_version</code>	Retrieve the server version from <code>Server_Info</code> , if the version is valid.
<code>get_wrapper_name</code>	Retrieve the wrapper name from <code>Server_Info</code> , if the name is valid.
<code>merge</code>	Merge a delta <code>Server_Info</code> object into the current object.
<code>copy</code>	Duplicate a <code>Server_Info</code> object.

Server_Info constructor

Purpose

Construct an empty `Server_Info` object.

Syntax

```
Server_Info ()
```

Input arguments

None.

Output arguments

None.

Return value

None.

Server_Info constructor

Purpose

Construct a `Server_Info` object with specified parameters.

Syntax

```
Server_Info (sqlint32* a_rc,  
            sqluint8* a_server_name,  
            sqlint32 a_name_len,  
            sqluint8* a_server_type,
```

```

sqlint32  a_type_len,
sqluint8* a_server_version,
sqlint32  a_ver_len,
sqluint8* a_server_wrapper,
sqlint32  a_wrap_len);

```

Input arguments

Table 20. Input arguments for the *Server_Info* constructor

Name	Data type	Description
a_server_name	sqluint8*	Server name (not null-terminated).
a_name_len	sqlint32	Length of the server name.
a_server_type	sqluint8*	Server type (not null-terminated).
a_type_len	sqlint32	Length of the server type.
a_server_version	sqluint8*	Server version (not null-terminated).
a_ver_len	sqlint32	Length of the server version.
a_server_wrapper	sqluint8*	Name of wrapper (not null-terminated).
a_wrap_len	sqlint32	Length of the wrapper name.

Output arguments

Table 21. Output arguments for the *Server_info* constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code; 0 indicates success.

Return value

None.

add_option function

Where defined

Catalog_Info

Purpose

Add a single-value option to the catalog information.

Usage The wrapper can invoke this member function when

wrapper-generated options are added to the catalog during CREATE SERVER or ALTER SERVER statement processing.

Syntax

```
sqlint32 add_option (sqluint8* a_opt_name,  
                    sqlint32 a_opt_name_len,  
                    sqluint8* a_opt_value,  
                    sqlint32 a_opt_value_len,  
                    Catalog_Option::Action a_act  
                    = Catalog_Option::sqlqg_None,  
                    char* a_opt_type = "")
```

Input arguments

Table 22. Input arguments for the add_option member function

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_opt_name_len	sqlint32	Length of the option name.
a_opt_value	sqluint8*	Option value (not null-terminated).
a_opt_value_len	sqlint32	Length of the option value.
a_act	Catalog_Option::Action	Action for this option (ADD, SET, DROP, or none).
a_opt_type	char*	Token that is used in the SQLN1884 error message if this is a duplicate option. Use the SQLQG_SERVER_OPTION constant that is defined in the sqlqg_misc.h header file.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

drop_option function

Where defined

Catalog_Info

Purpose

Delete an option from a Catalog_Info class. This member function does not drop the option from the catalog. The option is dropped when an option to a delta Catalog_Info object is added with an action of Catalog_Option::sqlqg_Drop.

Syntax


```
sqlint32 drop_option (Catalog_Option* a_option)
```

Input arguments

Table 23. Input arguments for the `drop_option` member function

Name	Data type	Description
<code>a_option</code>	<code>Catalog_Option*</code>	Option to be dropped.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_option function

Where defined

`Catalog_Info`

Purpose

Retrieve a catalog option by name.

Usage The output argument `option` is null if the option is not found.

Syntax

```
sqlint32 get_option (sqluint8*      a_opt_name,
                   sqlint32      a_name_len,
                   Catalog_Option** a_option)
```

Input arguments

Table 24. Input arguments for the `get_option` member function

Name	Data type	Description
<code>a_opt_name</code>	<code>sqluint8*</code>	Option name (not null-terminated).
<code>a_name_len</code>	<code>sqlint32</code>	Length of the option name.

Output arguments

Table 25. Output arguments for the `get_option` member function

Name	Data type	Description
<code>a_option</code>	<code>Catalog_Option**</code>	Found option.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the option was not found.

get_first_option function

Where defined

Catalog_Info

Purpose

Retrieve a pointer to the first option in the option chain.

Syntax

```
Catalog_Option* get_first_option ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the first option in the option chain. The value is null if the chain is empty.

get_next_option function

Where defined

Catalog_Info

Purpose

Retrieve the next option in the option chain.

Syntax

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

Input arguments

Table 26. Input arguments for the get_next_option member function

Name	Data type	Description
a_current_option	Catalog_Option*	Current option.

Output arguments

None.

Return value

Pointer to the next option in the chain. The value is null if at the end.

get_basic_info function

Purpose

Retrieve basic information from a Server_Info object. Unlike other

get_XXX member routines, the get_basic_info member function returns SQLQG_ERROR and logs an error if any of the data items are not valid.

Syntax

```
sqlint32 get_basic_info (sqluint8** a_server_name,
                        sqluint8** a_server_type,
                        sqluint8** a_server_version,
                        sqluint8** a_server_wrapper)
```

Input arguments

None.

Output arguments

Table 27. Output arguments for the get_basic_info member function

Name	Data type	Description
a_server_name	sqluint8**	Pointer to the null-terminated server name.
a_server_type	sqluint8**	Pointer to the null-terminated server type.
a_server_version	sqluint8**	Pointer to the null-terminated server version.
a_server_wrapper	sqluint8**	Pointer to the null-terminated wrapper name.

Return value

Return code. A value of 0 indicates success.

get_server_name function

Purpose

Retrieve the server name from Server_Info, if the name is valid.

Syntax

```
sqlint32 get_server_name (sqluint8** a_server_name)
```

Input arguments

None.

Output arguments

Table 28. Output arguments for the `get_server_name` member function

Name	Data type	Description
<code>a_server_name</code>	<code>sqluint8**</code>	Pointer to a null-terminated server name.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the server name is not set.

`set_server_type` function

Purpose

Set the server type for the server.

Usage The wrapper can invoke this member function during `CREATE SERVER` or `ALTER SERVER` statement processing to supply a default value when a default value is not present in the original statement.

Syntax

```
sqlint32 set_server_type (sqluint8* a_server_type,  
                        sqlint32 a_server_type_len)
```

Input arguments

Table 29. Input arguments for the `set_server_type` member function

Name	Data type	Description
<code>a_server_type</code>	<code>sqluint8*</code>	Server type (not null-terminated).
<code>a_server_type_len</code>	<code>sqlint32</code>	Length of server type.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

`get_server_type` function

Purpose

Retrieve the server type from `Server_Info`, if the type is valid.

Syntax

```
sqlint32 get_server_type (sqluint8** a_server_type)
```

Input arguments

None.

Output arguments*Table 30. Output arguments for the get_server_type member function*

Name	Data type	Description
a_server_type	sqluint8**	Pointer to a null-terminated server type.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the server type is not set.

set_server_version function**Purpose**

Set the server version for the server.

Usage The wrapper can invoke this member function during CREATE SERVER or ALTER SERVER statement processing to supply a default server version if a default server version is not specified in the DDL.

Syntax

```
sqlint32 set_server_version (sqluint8* a_server_version,
                             sqlint32 a_server_version_len)
```

Input arguments*Table 31. Input arguments for the set_server_version member function*

Name	Data type	Description
a_server_version	sqluint8*	Server version (not null-terminated).
a_server_version_len	sqlint32	Length of the server version.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Server_Info

get_server_version function

Purpose

Retrieve the server version from Server_Info, if the version is valid.

Syntax

```
sqlint32 get_server_version (sqluint8** a_server_version)
```

Input arguments

None.

Output arguments

Table 32. Output arguments for the get_server_version member function

Name	Data type	Description
a_server_version	sqluint8**	Pointer to a null-terminated server version.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the server version is not set.

get_wrapper_name function

Purpose

Retrieve the wrapper name from Server_Info, if the name is valid.

Syntax

```
sqlint32 get_wrapper_name (sqluint8** a_wrapper_name)
```

Input arguments

None.

Output arguments

Table 33. Output arguments for the get_wrapper_name member function

Name	Data type	Description
a_wrapper_name	sqluint8**	Pointer to a null-terminated wrapper name.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the wrapper name is not set.

merge function**Purpose**

Merge a delta Server_Info object into the current object.

Syntax

```
sqlint32 merge (Server_Info* a_delta_info)
```

Input arguments

Table 34. Input arguments for the merge member function

Name	Data type	Description
a_delta_info	Server_Info*	Delta object.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

copy function**Purpose**

Duplicate a Server_Info object.

Syntax

```
sqlint32 copy (Server_Info** a_new_server_info)
```

Input arguments

None.

Output arguments

Table 35. Output arguments for the copy member function

Name	Data type	Description
a_new_server_info	Server_Info**	Pointer to the new Server_Info object.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Catalog classes for the C++ API” on page 1

User_Info class

This topic describes the User_Info class and provides details for the constructor and the member functions.

Overview

The User_Info class encapsulates the catalog information for a user mapping from the CREATE USER MAPPING and ALTER USER MAPPING statements.

The User_Info class is one of the catalog classes for the C++ API.

Usage This class is instantiated by the DB2 federated server to contain information from a CREATE USER MAPPING or an ALTER USER MAPPING statement or to contain information from the DB2 Information Integrator catalog. This class is instantiated by the wrapper when information is added during CREATE USER MAPPING or ALTER USER MAPPING statement processing.

File sqlqg_catalog.h

Data members

None.

Constructors and member functions

The following tables describe the constructor and the member functions of the User_Info class. The constructor and functions are described in more detail after the tables.

Table 36. Constructors for the User_Info class

Constructor	Description
User_Info	Construct a default (empty) User_Info object.

Table 37. Member functions for the User_Info class

Member function	Description
get_server_name	Retrieve the server name for this user mapping.
get_authid	Retrieve the authorization ID for this user mapping.
merge	Merge a delta User_Info object into the current object.
copy	Create a copy of the current User_Info object.
add_option	Add a single-value option to the catalog information.

Table 37. Member functions for the User_Info class (continued)

Member function	Description
drop_option	Delete an option from a Catalog_Info class.
get_option	Retrieve a catalog option by name.
get_first_option	Retrieve a pointer to the first option in the option chain.
get_next_option	Retrieve the next option in the option chain.

User_Info constructor**Purpose**

Construct a default (empty) User_Info object.

Syntax

User_Info ()

Input arguments

None.

Output arguments

None.

Return value

None.

get_server_name function**Purpose**

Retrieve the server name for this user mapping.

Syntax

sqlint32 get_server_name (sqluint8** a_server_name)

Input arguments

None.

Output arguments

Table 38. Output arguments for the get_server_name member function

Name	Data type	Description
a_server_name	sqluint8**	Pointer to a null-terminated server name.

User_Info

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the server name is not valid or is not set.

get_authid function

Purpose

Retrieve the authorization ID for this user mapping.

Syntax

```
sqlint32 get_authid (sqluint8** a_authid)
```

Input arguments

None.

Output arguments

Table 39. Output arguments for the `get_authid` member function

Name	Data type	Description
<code>a_authid</code>	<code>sqluint8**</code>	Pointer to the null-terminated authorization ID.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the ID is not valid or is not set.

merge function

Purpose

Merge a delta `User_Info` object into the current object.

Syntax

```
sqlint32 merge (User_Info* a_delta_info)
```

Input arguments

Table 40. Input arguments for the `merge` member function

Name	Data type	Description
<code>a_delta_info</code>	<code>User_Info*</code>	Delta object to be merged.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

copy function**Purpose**

Create a copy of the current User_Info object.

Syntax

```
sqlint32 copy (User_Info** a_new_user_info)
```

Input arguments

None.

Output arguments

Table 41. Output arguments for the copy member function

Name	Data type	Description
a_new_user_info	User_Info**	Pointer to the duplicate object.

Return value

Return code. A value of 0 indicates success.

add_option function**Purpose**

Add a single-value option to the catalog information.

Usage The wrapper can invoke this member function during CREATE USER MAPPING or ALTER USER MAPPING statement processing to add wrapper-generated options.

Syntax

```
sqlint32 add_option (sqluint8* a_opt_name,
                    sqlint32 a_opt_name_len,
                    sqluint8* a_opt_value,
                    sqlint32 a_opt_value_len,
                    Catalog_Option::Action a_act
                    = Catalog_Option::sqlqg_None,
                    char* option_type = "")
```

Input arguments

Table 42. Input arguments for the add_option member function

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_opt_name_len	sqlint32	Length of the option name.
a_opt_value	sqluint8*	Option value (not null-terminated).

Table 42. Input arguments for the `add_option` member function (continued)

Name	Data type	Description
<code>a_opt_value_len</code>	<code>sqlint32</code>	Length of the option value.
<code>a_act</code>	<code>Catalog_Option::Action</code>	Action for this option (ADD, SET, DROP, or none).
<code>a_opt_type</code>	<code>char*</code>	Token that is used in an SQLN1884 error message if this is a duplicate option. Use the <code>SQLQG_USER_OPTION</code> constant that is defined in the <code>sqlqg_misc.h</code> header file.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

drop_option function

Where defined

`Catalog_Info`

Purpose

Delete an option from a `Catalog_Info` class. This member function does not drop the option from the catalog. The option drops when an option to a delta `Catalog_Info` object is added with an action of `Catalog_Option::sqlqg_Drop`.

Syntax

```
sqlint32 drop_option (Catalog_Option* a_option)
```

Input arguments

Table 43. Input arguments for the `drop_option` member function

Name	Data type	Description
<code>a_option</code>	<code>Catalog_Option*</code>	Option to be dropped.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_option function**Where defined**

Catalog_Info

Purpose

Retrieve a catalog option by name.

Usage The output argument option is null if the option is not found.**Syntax**

```
sqlint32 get_option (sqluint8*      a_opt_name,
                    sqlint32      a_name_len,
                    Catalog_Option** a_option)
```

Input arguments*Table 44. Input arguments for the get_option member function*

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_name_len	sqlint32	Length of the option name.

Output arguments*Table 45. Output arguments for the get_option member function*

Name	Data type	Description
a_option	Catalog_Option**	Found option.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the option was not found.

get_first_option function**Where defined**

Catalog_Info

Purpose

Retrieve a pointer to the first option in the option chain.

Syntax

```
Catalog_Option* get_first_option ()
```

Input arguments

None.

User_Info

Output arguments

None.

Return value

Pointer to the first option in the option chain. The value is null if the chain is empty.

get_next_option function

Where defined

Catalog_Info

Purpose

Retrieve the next option in the option chain.

Syntax

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

Input arguments

Table 46. Input arguments for the get_next_option member function

Name	Data type	Description
a_current_option	Catalog_Option*	Current option.

Output arguments

None.

Return value

Pointer to next option in the chain. The value is null if at the end.

Related reference:

- “Catalog classes for the C++ API” on page 1

Column_Info class

This topic describes the Column_Info class and provides details for the constructor and the member functions.

Overview

The Column_Info class encapsulates catalog information for a column of a nickname. This class includes column-statistical information.

The Column_Info class is one of the catalog classes for the C++ API.

Usage This class is instantiated by the DB2 federated server to contain information from a CREATE NICKNAME or an ALTER NICKNAME statement or to contain information from the DB2 Information

Integrator catalog. This class is instantiated by the wrapper when information is added during CREATE NICKNAME or ALTER NICKNAME statement operations.

File sqlqg_catalog.h

Data members

None.

Constructors and member functions

The following tables describe the constructor and the member functions of the Column_Info class. The constructor and functions are described in more detail after the tables.

Table 47. Constructors for the Column_Info class

Constructor	Description
Column_Info	Construct a default (empty) Column_Info object.

Table 48. Member functions for the Column_Info class

Member function	Description
set_column_name	Set the name for the column.
get_column_name	Retrieve the name for this column.
get_new_column_name	Retrieve the new column name that is specified in an ALTER COLUMN statement that includes an ALTER (or SET) COLUMN clause to rename the column.
set_column_type	Set the type for the column.
get_column_type	Retrieve the local column type from Server_Info if the type is valid.
set_for_bit_data	Set the FOR BIT DATA flag for the column.
get_for_bit_data	Retrieve the FOR BIT DATA flag for the column, if the flag is valid.
set_column_ID	Set the ID (position) for the column.
get_column_ID	Retrieve the ID (position) for the column, if the ID is valid.
set_org_length	Set the maximum length for the column.
get_org_length	Retrieve the maximum column length from Column_Info, if the value is valid.
set_org_scale	Set the numeric scale for the column.

Column_Info

Table 48. Member functions for the Column_Info class (continued)

Member function	Description
get_org_scale	Retrieve the numeric scale from Column_Info, if the value is valid.
set_nulls	Set the nulls-allowed flag for the column.
get_nulls	Retrieve the nulls-allowed flag from Column_Info, if the value is valid.
set_avg_length	Set the average length for the column.
get_avg_length	Retrieve the average column length from Column_Info, if the value is valid.
set_high2key	Set the second-highest value for the column.
get_high2key	Retrieve the second-highest value from Column_Info, if the value is valid.
set_low2key	Set the second-lowest value for the column.
get_low2key	Retrieve the second-highest value from Column_Info, if the value is valid.
get_default	Retrieve the default value from Column_Info, if the value is valid.
set_colcard	Set the cardinality for the column.
get_colcard	Retrieve the cardinality from Column_Info, if the value is valid.
set_codepage1	Set the code page for the column.
get_codepage1	Retrieve the code page from Column_Info, if the value is valid.
set_codepage2	Set the code page for the column.
get_codepage2	Retrieve the code page from Column_Info, if the value is valid.
merge	Merge a delta Column_Info object into the current object.
copy	Duplicate a Column_Info object.
add_option	Add a single-value option to the catalog information.
drop_option	Delete an option from a Catalog_Info class.
get_option	Retrieve a catalog option by name.
get_first_option	Retrieve a pointer to the first option in the option chain.

Table 48. Member functions for the Column_Info class (continued)

Member function	Description
get_next_option	Retrieve the next option in the option chain.

Column_Info constructor**Purpose**

Construct a default (empty) Column_Info object.

Syntax

```
Column_Info ()
```

Input arguments

None.

Output arguments

None.

Return value

None.

set_column_name function**Purpose**

Set the name for the column.

Usage Do not use this member function to change the name of a column.

Syntax

```
sqlint32 set_column_name (sqluint8* a_column_name,
                          sqlint32 a_column_name_len)
```

Input arguments

Table 49. Input arguments for the set_column_name member function

Name	Data type	Description
a_column_name	sqluint8*	Column name (not null-terminated).
a_column_name_len	sqlint32	Length of the column name.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_column_name function

Purpose

Retrieve the name for this column.

Syntax

```
sqlint32 get_column_name (sqluint8** a_column_name)
```

Input arguments

None.

Output arguments

Table 50. Output arguments for the get_column_name member function

Name	Data type	Description
a_column_name	sqluint8**	Pointer to a null-terminated column name.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the column name is not valid or is not set.

get_new_column_name function

Purpose

Retrieve the new column name that is specified in an ALTER COLUMN statement that includes an ALTER (or SET) COLUMN clause to rename the column.

Syntax

```
sqlint32 get_new_column_name (sqluint8** a_new_col_name)
```

Input arguments

None.

Output arguments

Table 51. Output arguments for the get_new_column_name member function

Name	Data type	Description
a_new_col_name	sqluint8**	Pointer to a new null-terminated column name.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the new column name is not set.

set_column_type function**Purpose**

Set the type for the column.

Syntax

```
sqlint32 set_column_type (sqluint8* a_column_type,
                        sqlint32 a_column_type_len)
```

Input arguments

Table 52. Input arguments for the set_column_type member function

Name	Data type	Description
a_column_type	sqluint8*	Column type (not null-terminated).
a_column_type_len	sqlint32	Length of the column type.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_column_type function**Purpose**

Retrieve the local column type from Server_Info, if the type is valid.

Syntax

```
sqlint32 get_column_type (sqluint8** a_column_type)
```

Input arguments

None.

Output arguments

Table 53. Output arguments for the get_column_type member function

Name	Data type	Description
a_column_type	sqluint8**	Pointer to the null-terminated column type.

Column_Info

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the type is not set.

set_for_bit_data function

Purpose

Set the FOR BIT DATA flag for the column.

Syntax

```
void set_for_bit_data (sqluint8 a_for_bit_data)
```

Input arguments

Table 54. Input arguments for the set_for_bit_data member function

Name	Data type	Description
a_for_bit_data	sqluint8	FOR BIT DATA flag (the flag must be 'Y' or 'N').

Output arguments

None.

Return value

None.

get_for_bit_data function

Purpose

Retrieve the FOR BIT DATA flag for the column, if the flag is valid.

Syntax

```
sqlint32 get_for_bit_data (sqluint8* a_for_bit_data)
```

Input arguments

None.

Output arguments

Table 55. Output arguments for the get_for_bit_data member function

Name	Data type	Description
a_for_bit_data	sqluint8*	The flag, which must be 'Y' or 'N'.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the flag is not set.

set_column_ID function**Purpose**

Set the ID (position) for the column.

Usage Do not use this member function to change the order of the columns in a nickname.

Syntax

```
sqlint32 set_column_ID (sqlint16 a_column_ID)
```

Input arguments

Table 56. Input arguments for the set_column_ID member function

Name	Data type	Description
a_column_ID	sqlint16	Column ID (position).

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_column_ID function**Purpose**

Retrieve the ID (position) for the column, if the ID is valid.

Syntax

```
sqlint32 get_column_ID (sqluint16* a_column_ID)
```

Input arguments

None.

Output arguments

Table 57. Output arguments for the get_column_ID member function

Name	Data type	Description
a_column_ID	sqluint16*	Column ID.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the ID is not set.

Column_Info

set_org_length function

Purpose

Set the maximum length (in bytes) for the column.

Syntax

```
void set_org_length (sqlint32 a_org_length)
```

Input arguments

Table 58. Input arguments for the set_org_length member function

Name	Data type	Description
a_org_length	sqlint32	Maximum length of the column.

Output arguments

None.

Return value

None.

get_org_length function

Purpose

Retrieve the maximum column length (in bytes) from Column_Info, if the value is valid.

Syntax

```
sqlint32 get_org_length (sqlint32* a_org_length)
```

Input arguments

None.

Output arguments

Table 59. Output arguments for the get_org_length member function

Name	Data type	Description
a_org_length	sqlint32*	Maximum length of the column.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_org_scale function**Purpose**

Set the numeric scale for the column.

Syntax

```
void set_org_scale (sqlint16 a_org_scale)
```

Input arguments

Table 60. Input arguments for the set_org_scale member function

Name	Data type	Description
a_org_scale	sqlint16	Scale.

Output arguments

None.

Return value

None.

get_org_scale function**Purpose**

Retrieve the numeric scale from Column_Info, if the value is valid.

Syntax

```
sqlint32 get_org_scale (sqlint16* a_org_scale)
```

Input arguments

None.

Output arguments

Table 61. Output arguments for the get_org_scale member function

Name	Data type	Description
a_org_scale	sqlint16*	Numeric scale.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_nulls function**Purpose**

Set the nulls-allowed flag for the column.

Syntax

Column_Info

```
void set_nulls (sqluint8 a_nulls)
```

Input arguments

Table 62. Input arguments for the `set_nulls` member function

Name	Data type	Description
<code>a_nulls</code>	<code>sqluint8</code>	Nulls-allowed flag. The flag must be 'Y' or 'N'.

Output arguments

None.

Return value

None.

get_nulls function

Purpose

Retrieve the nulls-allowed flag from `Column_Info`, if the value is valid.

Syntax

```
sqlint32 get_nulls (sqluint8* a_nulls)
```

Input arguments

None.

Output arguments

Table 63. Output arguments for the `get_nulls` member function

Name	Data type	Description
<code>a_nulls</code>	<code>sqluint8*</code>	Nulls-allowed flag of the column.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the value is not set.

set_avg_length function

Purpose

Set the average length (in bytes) for the column.

Usage The wrapper sets the average length of a column during `CREATE NICKNAME` or `ALTER NICKNAME` statement processing. The DB2

optimizer uses this average length information when the optimizer develops a query optimization plan.

Syntax

```
void set_avg_len (sqlint32 a_avg_len)
```

Input arguments

Table 64. Input arguments for the set_avg_length member function

Name	Data type	Description
a_avg_len	sqlint32	Average length of the column.

Output arguments

None.

Return value

None.

get_avg_length function

Purpose

Retrieve the average column length (in bytes) from Column_Info, if the value is valid.

Syntax

```
sqlint32 get_avg_length (sqlint32* a_avg_len)
```

Input arguments

None.

Output arguments

Table 65. Output arguments for the get_avg_length member function

Name	Data type	Description
a_avg_len	sqlint32*	Average length of the column.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_high2key function

Purpose

Set the second-highest value for the column.

Column_Info

Usage The wrapper can set the second-highest value of a column during CREATE NICKNAME or ALTER NICKNAME statement processing. The DB2 optimizer can use this second-highest value or the highest value when the optimizer develops a query optimization plan.

Syntax

```
sqlint32 set_high2key (sqluint8* a_high2key,  
                      sqlint32 a_high2key_len)
```

Input arguments

Table 66. Input arguments for the set_high2key member function

Name	Data type	Description
a_high2key	sqluint8*	Second-highest character string value (not null-terminated).
a_high2key_len	sqlint32	Length of the value.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_high2key function

Purpose

Retrieve the second-highest value from Column_Info, if the value is valid.

Syntax

```
sqlint32 get_high2key (sqluint8** a_high2key)
```

Input arguments

None.

Output arguments

Table 67. Output arguments for the get_high2key member function

Name	Data type	Description
a_high2key	sqluint8**	Second-highest value of the column.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_low2key function**Purpose**

Set the second-lowest value for the column.

Usage The wrapper can set the second-lowest value of a column during CREATE NICKNAME or ALTER NICKNAME statement processing. The DB2 optimizer can use this second-lowest value or the lowest value when the optimizer develops a query optimization plan.

Syntax

```
sqlint32 set_low2key (sqluint8* a_low2key,
                    sqlint32 a_low2key_len)
```

Input arguments

Table 68. Input arguments for the set_low2key member function

Name	Data type	Description
a_low2key	sqluint8*	Second-lowest character string value (not null-terminated).
a_low2key_len	sqlint32	Length of the value.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_low2key function**Purpose**

Retrieve the second-highest value from Column_Info, if the value is valid.

Syntax

```
sqlint32 get_low2key (sqluint8** a_low2key)
```

Input arguments

None.

Output arguments

Table 69. Output arguments for the `get_low2key` member function

Name	Data type	Description
<code>a_low2key</code>	<code>sqluint8**</code>	Second-lowest value of the column.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the value is not set.

`get_default` function

Purpose

Retrieve the default value from `Column_Info`, if the value is valid.

Syntax

```
sqlint32 get_default (sqluint8** a_default)
```

Input arguments

None.

Output arguments

Table 70. Output arguments for the `get_default` member function

Name	Data type	Description
<code>a_default</code>	<code>sqluint8**</code>	Default value of the column.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the value is not set.

`set_colcard` function

Purpose

Set the cardinality for the column.

Usage The wrapper sets the column cardinality (if known) during `CREATE NICKNAME` or `ALTER NICKNAME` statement processing. The DB2 optimizer uses this information when it generates an optimal performance plan. For columns with distinct values (no duplicates), the column cardinality must be the same as the nickname cardinality. The DB2 optimizer generates an error if the column cardinality is greater than the nickname cardinality.

Syntax

```
void set_colcard (sqlint64 a_colcard)
```

Input arguments*Table 71. Input arguments for the set_colcard member function*

Name	Data type	Description
a_colcard	sqlint64	Cardinality of the column.

Output arguments

None.

Return value

None.

get_colcard function**Purpose**

Retrieve the cardinality from Column_Info, if the value is valid.

Syntax

```
sqlint32 get_colcard (sqlint64* a_colcard)
```

Input arguments

None.

Output arguments*Table 72. Output arguments for the get_colcard member function*

Name	Data type	Description
a_colcard	sqlint64*	Cardinality of the column.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_codepage1 function**Purpose**

Set the code page for the column.

Syntax

```
void set_codepage1 (sqlint16 a_codepage1)
```

Column_Info

Input arguments

Table 73. Input arguments for the `set_codepage1` member function

Name	Data type	Description
<code>a_codepage1</code>	<code>sqlint16</code>	Code page of the column.

Output arguments

None.

Return value

None.

`get_codepage1` function

Purpose

Retrieve the code page from `Column_Info`, if the value is valid.

Syntax

```
sqlint32 get_codepage1 (sqlint16* a_codepage1)
```

Input arguments

None.

Output arguments

Table 74. Output arguments for the `get_codepage1` member function

Name	Data type	Description
<code>a_codepage1</code>	<code>sqlint16*</code>	Code page of the column.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the value is not set.

`set_codepage2` function

Purpose

Set the code page for the column.

Syntax

```
void set_codepage2 (sqlint16 a_codepage2)
```

Input arguments*Table 75. Input arguments for the set_codepage2 member function*

Name	Data type	Description
a_codepage2	sqlint16	Code page of the column.

Output arguments

None.

Return value

None.

get_codepage2 function**Purpose**

Retrieve the code page from Column_Info, if the value is valid.

Syntax

sqlint32 get_codepage2 (sqlint16* a_codepage2)

Input arguments

None.

Output arguments*Table 76. Output arguments for the get_codepage2 member function*

Name	Data type	Description
a_codepage2	sqlint16*	Code page of the column.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

merge function**Purpose**

Merge a delta Column_Info object into the current object.

Syntax

sqlint32 merge (Column_Info* a_delta_info)

Column_Info

Input arguments

Table 77. Input arguments for the merge member function

Name	Data type	Description
a_delta_info	Column_Info*	Data to be merged.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

copy function

Purpose

Duplicate a Column_Info object.

Syntax

```
sqlint32 copy (Column_Info** a_new_column_info)
```

Input arguments

None.

Output arguments

Table 78. Output arguments for the copy member function

Name	Data type	Description
a_new_column_info	Column_Info**	Pointer to the duplicate object.

Return value

Return code. A value of 0 indicates success.

add_option function

Where defined

Catalog_Info

Purpose

Add a single-value option to the catalog information.

Usage The wrapper invokes this member function when wrapper-generated options are added to the catalog during CREATE NICKNAME or ALTER NICKNAME statement processing.

Syntax


```

sqlint32 add_option (sqluint8*  a_opt_name,
                    sqlint32   a_opt_name_len,
                    sqluint8*  a_opt_value,
                    sqlint32   a_opt_value_len,
                    Catalog_Option::Action a_act
                    = Catalog_Option::sqlqg_None,
                    char*      a_opt_type = "")

```

Input arguments

Table 79. Input arguments for the add_option member function

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_opt_name_len	sqlint32	Length of the option name.
a_opt_value	sqluint8*	Option value (not null-terminated).
a_opt_value_len	sqlint32	Length of the option value.
a_act	Catalog_Option::Action	Action for this option (ADD, SET, DROP, or none).
a_opt_type	char*	Token that is used in the SQLN1884 error message if this is a duplicate option. Use the SQLQG_COLUMN_OPTION constant that is defined in the sqlqg_misc.h header file.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

drop_option function

Where defined

Column_Info

Purpose

Delete an option from a Column_Info class. This member function does not drop the option from the catalog. The option drops when an option to a delta Catalog_Info object is added with an action of Catalog_Option::sqlqg_Drop.

Syntax

```

sqlint32 drop_option (Catalog_Option* a_option)

```

Input arguments

Table 80. Input arguments for the drop_option member function

Name	Data type	Description
a_option	Catalog_Option*	Option to be dropped.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_option function

Where defined

Catalog_Info

Purpose

Retrieve a catalog option by name.

Usage The output argument a_option is null if the option is not found.

Syntax

```
sqlint32 get_option (sqluint8*      a_opt_name,
                    sqlint32      a_name_len,
                    Catalog_Option** a_option)
```

Input arguments

Table 81. Input arguments for the get_option member function

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_name_len	sqlint32	Length of the option name.

Output arguments

Table 82. Output arguments for the get_option member function

Name	Data type	Description
a_option	Catalog_Option**	Found option.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the option was not found.

get_first_option function**Where defined**

Catalog_Info

Purpose

Retrieve a pointer to the first option in the option chain.

Syntax

Catalog_Option* get_first_option ()

Input arguments

None.

Output arguments

None.

Return value

Pointer to the first option in the chain. The value is null if the chain is empty.

get_next_option function**Where defined**

Catalog_Info

Purpose

Retrieve the next option in the option chain.

Syntax

Catalog_Option* get_next_option (Catalog_Option* a_current_option)

Input arguments*Table 83. Input arguments for the get_next_option member function*

Name	Data type	Description
a_current_option	Catalog_Option*	Current option.

Output arguments

None.

Return value

Pointer to the next option in the chain. The value is null if at the end.

Related reference:

- “Catalog classes for the C++ API” on page 1

Nickname_Info

Nickname_Info class

This topic describes the Nickname_Info class and provides details for the constructor and the member functions.

Overview

The Nickname_Info class encapsulates a nickname definition from the catalog and includes column definitions.

The Nickname_Info class is one of the catalog classes for the C++ API.

Usage This class is instantiated by the DB2 federated server to contain information from a CREATE NICKNAME or an ALTER NICKNAME statement or to contain information from the DB2 Information Integrator catalog. This class is instantiated by the wrapper when information is added during CREATE NICKNAME or ALTER NICKNAME statement operations.

File sqlqg_catalog.h

Data members

None.

Constructors and member functions

The following tables describe the constructor and the member functions of the Nickname_Info class. The constructor and functions are described in more detail after the tables.

Table 84. Constructors for the Nickname_Info class

Constructor	Description
Nickname_Info	Construct a default (empty) Nickname_Info object.

Table 85. Member functions for the Nickname_Info class

Member function	Description
get_local_schema	Retrieve the local schema name for the nickname.
get_nickname	Retrieve the local name for the nickname.
get_server_name	Retrieve the server name for the nickname.
set_card	Set the cardinality for the nickname.
get_card	Retrieve the nickname cardinality, if the value is valid.
set_npages	Set the npages value for the nickname.

Table 85. Member functions for the Nickname_Info class (continued)

Member function	Description
get_npages	Retrieve the nickname npages statistic, if the value is valid.
set_fpages	Set the fpages statistic for the nickname.
get_fpages	Retrieve the nickname fpages statistic, if the value is valid.
set_overflow	Set the overflow statistic for the nickname.
get_overflow	Retrieve the nickname overflow statistic, if the value is valid.
insert_column	Add a column definition to the nickname information.
replace_column	Replace a column definition in the nickname information.
get_number_columns	Retrieve the number of columns for the nickname.
get_first_column	Retrieve a pointer to the first Column_Info object for the nickname.
get_next_column	Retrieve a pointer to the next Column_Info object.
get_column	Retrieve a column description by (local) name.
merge	Merge a delta Nickname_Info object into the current object.
copy	Duplicate a Nickname_Info object.
add_option	Add a single-value option to the catalog information.
drop_option	Delete an option from a Catalog_Info class.
get_option	Retrieve a catalog option by name.
get_first_option	Retrieve a pointer to the first option in the option chain.
get_next_option	Retrieve the next option in the option chain.

Nickname_Info constructor

Purpose

Construct a default (empty) Nickname_Info object.

Syntax

Nickname_Info

Nickname_Info ()

Input arguments

None.

Output arguments

None.

Return value

None.

get_local_schema function

Purpose

Retrieve the local schema name for the nickname.

Syntax

```
sqlint32 get_local_schema (sqluint8** a_local_schema)
```

Input arguments

None.

Output arguments

Table 86. Output arguments for the get_local_schema member function

Name	Data type	Description
a_local_schema	sqluint8**	Pointer to the null-terminated local schema name.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

get_nickname function

Purpose

Retrieve the local name for the nickname.

Syntax

```
sqlint32 get_nickname (sqluint8** a_nickname)
```

Input arguments

None.

Output arguments*Table 87. Output arguments for the get_nickname member function*

Name	Data type	Description
a_nickname	sqluint8**	Pointer to the null-terminated local name.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

get_server_name function**Purpose**

Retrieve the server name for the nickname.

Syntax

```
sqlint32 get_server_name (sqluint8** a_server_name)
```

Input arguments

None.

Output arguments*Table 88. Output arguments for the get_server_name member function*

Name	Data type	Description
a_server_name	sqluint8**	Pointer to the null-terminated server name.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_card function**Purpose**

Set the cardinality for the nickname.

Usage The wrapper can invoke this member function during CREATE NICKNAME or ALTER NICKNAME statement processing to specify an initial cardinality for a nickname. The set_colcard method of the Column_Info object can be used to set individual column cardinalities.

Nickname_Info

For columns with unique values, the column cardinality must equal the nickname cardinality. The column cardinality cannot be greater than the nickname cardinality.

Syntax

```
void set_card (sqlint64 a_card)
```

Input arguments

Table 89. Input arguments for the set_card member function

Name	Data type	Description
a_card	sqlint64	Cardinality for the nickname.

Output arguments

None.

Return value

None.

get_card function

Purpose

Retrieve the nickname cardinality, if the value is valid.

Syntax

```
sqlint32 get_card (sqlint64* a_card)
```

Input arguments

None.

Output arguments

Table 90. Output arguments for the get_card member function

Name	Data type	Description
a_card	sqlint64*	Nickname cardinality.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_npages function

Purpose

Set the npages value for the nickname.

Syntax

```
void set_npages (sqlint32 a_npages)
```

Input arguments

Table 91. Input arguments for the set_npages member function

Name	Data type	Description
a_npages	sqlint32	Npages value for the nickname.

Output arguments

None.

Return value

None.

get_npages function**Purpose**

Retrieve the nickname npages statistic, if the value is valid.

Syntax

```
sqlint32 get_npages (sqlint32* a_npages)
```

Input arguments

None.

Output arguments

Table 92. Output arguments for the get_npages member function

Name	Data type	Description
a_npages	sqlint32*	Nickname npages statistic.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

set_fpages function**Purpose**

Set the fpages statistic for the nickname.

Syntax

```
void set_fpages (sqlint32 a_fpages)
```

Nickname_Info

Input arguments

Table 93. Input arguments for the `set_fpages` member function

Name	Data type	Description
<code>a_fpages</code>	<code>sqlint32</code>	Fpages statistic for the nickname.

Output arguments

None.

Return value

None.

`get_fpages` function

Purpose

Retrieve the nickname fpages statistic, if the value is valid.

Syntax

```
sqlint32 get_fpages (sqlint32* a_fpages)
```

Input arguments

None.

Output arguments

Table 94. Output arguments for the `get_fpages` member function

Name	Data type	Description
<code>a_fpages</code>	<code>sqlint32*</code>	Nickname fpages statistic.

Return value

Return code. A value of 0 indicates success. `SQLQG_NOVALUE` indicates that the value is not set.

`set_overflow` function

Purpose

Set the overflow statistic for the nickname.

Syntax

```
void set_overflow (sqlint32 a_overflow)
```

Input arguments*Table 95. Input arguments for the set_overflow member function*

Name	Data type	Description
a_overflow	sqlint32	Overflow statistic for the nickname.

Output arguments

None.

Return value

None.

get_overflow function**Purpose**

Retrieve the nickname overflow statistic, if the value is valid.

Syntax

sqlint32 get_overflow (sqlint32* a_overflow)

Input arguments

None.

Output arguments*Table 96. Output arguments for the get_overflow member function*

Name	Data type	Description
a_overflow	sqlint32*	Nickname overflow statistic.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the value is not set.

insert_column function**Purpose**

Add a column definition to the nickname information.

Usage The Column_Info object must be allocated on the heap (using new). This routine takes control of the pointer. If the column ID for the a_new_col_info argument is set, an error occurs.

Syntax

sqlint32 insert_column (Column_Info* a_new_col_info)

Nickname_Info

Input arguments

Table 97. Input arguments for the `insert_column` member function

Name	Data type	Description
<code>a_new_col_info</code>	<code>Column_Info*</code>	<code>Column_Info</code> object to add.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

replace_column function

Purpose

Replace a column definition in the nickname information. Two column definitions are considered the same if these column definitions use the same ID.

Usage If the column ID is not set in the `a_new_col_info` argument, an error occurs.

Syntax

```
sqlint32 replace_column (Column_Info* a_new_col_info)
```

Input arguments

Table 98. Input arguments for the `replace_column` member function

Name	Data type	Description
<code>a_new_col_info</code>	<code>Column_Info*</code>	<code>Column_Info</code> to replace.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_number_columns function

Purpose

Retrieve the number of columns for the nickname.

Syntax

```
sqlint16 get_number_columns ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of columns.

get_first_column function**Purpose**

Retrieve a pointer to the first Column_Info object for the nickname.

Usage Columns are maintained in order by ID.**Syntax**

Column_Info* get_first_column ()

Input arguments

None.

Output arguments

None.

Return value

Pointer to first Column_Info object. The value is null if the list is empty.

get_next_column function**Purpose**

Retrieve a pointer to the next Column_Info object.

Usage Columns are maintained in order by ID.**Syntax**

Column_Info* get_next_column (Column_Info* a_cur_col_info)

Input arguments*Table 99. Input arguments for the get_next_column member function*

Name	Data type	Description
a_cur_col_info	Column_Info*	Current column.

Output arguments

None.

Nickname_Info

Return value

Pointer to next Column_Info. The value is null if at the end of the list.

get_column function

Purpose

Retrieve a column description by (local) name.

Syntax

```
sqlint32 get_column (sqluint8*   a_col_name,  
                    sqlint32    a_col_name_len,  
                    Column_Info** a_col_info)
```

Input arguments

Table 100. Input arguments for the get_column member function

Name	Data type	Description
a_col_name	sqluint8*	Column name (not null-terminated).
a_col_name_len	sqlint32	Length of the column name.

Output arguments

Table 101. Output arguments for the get_column member function

Name	Data type	Description
a_col_info	Column_Info**	Pointer to the Column_Info object.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the column name is not found.

merge function

Purpose

Merge a delta Nickname_Info object into the current object.

Syntax

```
sqlint32 merge (Nickname_Info* a_delta_info)
```

Input arguments*Table 102. Input arguments for the merge member function*

Name	Data type	Description
a_delta_info	Nickname_Info*	Data to be merged.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

copy function**Purpose**

Duplicate a Nickname_Info object.

Syntax

```
sqlint32 copy (Nickname_Info** a_new_idx_info)
```

Input arguments

None.

Output arguments*Table 103. Output arguments for the copy member function*

Name	Data type	Name	Description
a_new_idx_info	Nickname_Info**		Pointer to the duplicate object.

Return value

Return code. A value of 0 indicates success.

add_option function**Where defined**

Catalog_Info

Purpose

Add a single-value option to the catalog information.

Usage The wrapper can invoke this member function when wrapper-generated options are added to the catalog during CREATE NICKNAME or ALTER NICKNAME statement processing.

Syntax

Nickname_Info

```
sqlint32 add_option (sqluint8*  a_opt_name,  
                    sqlint32   a_opt_name_len,  
                    sqluint8*  a_opt_value,  
                    sqlint32   a_opt_value_len,  
                    Catalog_Option::Action a_act  
                    = Catalog_Option::sqlqg_None,  
                    char*      a_opt_type = "")
```

Input arguments

Table 104. Input arguments for the `add_option` member function

Name	Data type	Description
<code>a_opt_name</code>	<code>sqluint8*</code>	Option name (not null-terminated).
<code>a_opt_name_len</code>	<code>sqlint32</code>	Length of the option name.
<code>a_opt_value</code>	<code>sqluint8*</code>	Option value (not null-terminated).
<code>a_opt_value_len</code>	<code>sqlint32</code>	Length of the option value.
<code>a_act</code>	<code>Catalog_Option::Action</code>	Action for this option (ADD, SET, DROP, or none).
<code>a_opt_type</code>	<code>char*</code>	Token to be used in the SQLN1884 error message if this is a duplicate option. Use the <code>SQLQG_NICKNAME_OPTION</code> constant that is defined in the <code>sqlqg_misc.h</code> header file.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

drop_option function

Where defined

`Catalog_Info`

Purpose

Delete an option from a `Catalog_Info` class.

Usage This member function does not drop the option from the catalog. The option is dropped from the catalog when an option to a delta `Catalog_Info` object is added with an action of `Catalog_Option::sqlqg_Drop`.

Syntax

```
sqlint32 drop_option (Catalog_Option* a_option)
```


Input arguments*Table 105. Input arguments for the drop_option member function*

Name	Data type	Description
a_option	Catalog_Option*	Option to be dropped.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_option function**Where defined**

Catalog_Info

Purpose

Retrieve a catalog option by name.

Usage The output argument a_option is null if the option is not found.**Syntax**

```
sqlint32 get_option (sqluint8*      a_opt_name,
                    sqlint32      a_name_len,
                    Catalog_Option** a_option)
```

Input arguments*Table 106. Input arguments for the get_option member function*

Name	Data type	Description
a_opt_name	sqluint8*	Option name (not null-terminated).
a_name_len	sqlint32	Length of the option name.

Output arguments*Table 107. Output arguments for the get_option member function*

Name	Data type	Description
a_option	Catalog_Option**	Found option.

Return value

Return code. A value of 0 indicates success. SQLQG_NOVALUE indicates that the option was not found.

Nickname_Info

get_first_option function

Where defined

Catalog_Info

Purpose

Retrieve a pointer to the first option in the option chain.

Syntax

```
Catalog_Option* get_first_option ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to first option in the chain. The value is null if the chain is empty.

get_next_option function

Where defined

Catalog_Info

Purpose

Retrieve the next option in the option chain.

Syntax

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

Input arguments

Table 108. Input arguments for the get_next_option member function

Name	Data type	Description
a_current_option	Catalog_Option*	Current option.

Output arguments

None.

Return value

Pointer to next option in the chain. The value is null if at the end.

Related reference:

- “Catalog classes for the C++ API” on page 1

Wrapper classes for the C++ API

The following table describes each wrapper class for the C++ API.

Table 109. Wrapper classes

Class name	Description
Unfenced_Generic_Wrapper	This class represents the wrapper on the unfenced (trusted) process space and is responsible for the verification of catalog data.
Fenced_Generic_Wrapper	This class represents the wrapper on the fenced process space.

Related reference:

- “Unfenced_Generic_Wrapper class” on page 67
- “Fenced_Generic_Wrapper class” on page 75

Unfenced_Generic_Wrapper class

This topic describes the Unfenced_Generic_Wrapper class and provides details for the constructor, the destructor, and the member functions.

Overview

The Unfenced_Generic_Wrapper class represents the wrapper on the unfenced (trusted) process space and verifies catalog data.

The Unfenced_Generic_Wrapper class is one of the wrapper classes for the C++ API.

Usage The wrapper must implement a subclass of the Unfenced_Generic_Wrapper class. This class is instantiated by the wrapper-specific UnfencedWrapper_Hook function.

File sqlqg_unfenced_generic_wrapper.h

Data members

The following table lists the data members that you can use with the Unfenced_Generic_Wrapper class.

Table 110. Data members for the Unfenced_Generic_Wrapper class

Name	Data type	Description
m_info	Wrapper_Info*	Contains the catalog information for the wrapper.

Unfenced_Generic_Wrapper

Table 110. Data members for the Unfenced_Generic_Wrapper class (continued)

Name	Data type	Description
m_name	sqluint8*	Pointer to the null-terminated character string that contains the name of the wrapper.
m_corelib	sqluint8*	Pointer to the null-terminated character string that contains the name of the wrapper library. This name is the wrapper library that is used in the CREATE WRAPPER statement. This name does not correspond to the library that is used during statement processing.
m_version	sqlint32	Version of the wrapper code.

Constructors, destructors, and member functions

The following tables describe the constructor, the destructor, and the member functions of the Unfenced_Generic_Wrapper class. The constructor, destructor, and functions are described in more detail after the tables.

Table 111. Constructors for the Unfenced_Generic_Wrapper class

Constructor	Description
Unfenced_Generic_Wrapper	Construct an instance of the Unfenced_Generic_Wrapper class.

Table 112. Destructors for the Unfenced_Generic_Wrapper class

Destructor	Description
~Unfenced_Generic_Wrapper	Destructor for the Unfenced_Generic_Wrapper class.

Table 113. Member functions for the Unfenced_Generic_Wrapper class

Member function	Description
verify_my_register_wrapper_info	Verify the catalog information when the CREATE WRAPPER statement is submitted.
verify_my_alter_wrapper_info	Verify the catalog information when the ALTER WRAPPER statement is submitted.

Table 113. Member functions for the Unfenced_Generic_Wrapper class (continued)

Member function	Description
get_name	Return a pointer to the null-terminated character string that contains the wrapper name.
get_corelib	Return a pointer to the null-terminated character string that contains the wrapper library name.
get_version	Return the version of the wrapper code.
get_info	Return a pointer to the wrapper catalog information object.
initialize_my_wrapper	Initialize the wrapper object state from the catalog information object.
create_server	Instantiate the appropriate subclass of Server for the wrapper.

Unfenced_Generic_Wrapper constructor

Purpose

Construct an instance of Unfenced_Generic_Wrapper.

Syntax

```
Unfenced_Generic_Wrapper (sqlint32* a_rc,
                          sqlint32 a_wrapper_version = 0)
```

Input arguments

Table 114. Input arguments for the Unfenced_Generic_Wrapper constructor

Name	Data type	Description
a_wrapper_version	sqlint32	Version of the wrapper code.

Output arguments

Table 115. Output arguments for the Unfenced_Generic_Wrapper constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return. The value must be 0 for success.

Return value

None.

Unfenced_Generic_Wrapper

~Unfenced_Generic_Wrapper destructor

Purpose

Destructor for the Unfenced_Generic_Wrapper class.

Syntax

```
~Unfenced_Generic_Wrapper ()
```

Input arguments

None.

Output arguments

None.

Return value

None.

verify_my_register_wrapper_info function

Purpose

Verify the catalog information when the CREATE WRAPPER statement is submitted.

Usage This member function can be implemented by the wrapper in the wrapper-specific subclass of Unfenced_Generic_Wrapper if wrapper-specific wrapper options are supported.

The wrapper checks whether a delta object was allocated before allocating one itself.

Syntax

```
virtual sqlint32 verify_my_register_wrapper_info  
(Wrapper_Info* a_wrapper_info,  
 Wrapper_Info** a_delta_info)
```

Input arguments

Table 116. Input arguments for the verify_my_register_wrapper_info member function

Name	Data type	Description
a_wrapper_info	Wrapper_Info*	Information from the CREATE WRAPPER statement.

Output arguments

Table 117. Output arguments for the verify_my_register_wrapper_info member function

Name	Data type	Description
a_delta_info	Wrapper_Info**	Information that is added by verify_my.

Return value

Return code. A value of 0 indicates success.

verify_my_alter_wrapper_info function**Purpose**

Verify the catalog information when the ALTER WRAPPER statement is submitted.

Usage This member function can be implemented by the wrapper in the wrapper-specific subclass of Unfenced_Generic_Wrapper if wrapper-specific wrapper options are supported.

The wrapper checks whether a delta object was allocated before allocating one itself.

Syntax

```
virtual sqlint32 verify_my_alter_wrapper_info
    (Wrapper_Info* a_wrapper_info,
     Wrapper_Info** a_delta_info)
```

Input arguments

Table 118. Input arguments for the verify_my_alter_wrapper_info member function

Name	Data type	Description
a_wrapper_info	Wrapper_Info*	Information from the ALTER WRAPPER statement.

Output arguments

Table 119. Output arguments for the verify_my_alter_wrapper_info member function

Name	Data type	Description
a_delta_info	Wrapper_Info**	Information added by verify_my.

Return value

Return code. A value of 0 indicates success.

get_name function**Where defined**

Wrapper

Unfenced_Generic_Wrapper

Purpose

Return a pointer to the null-terminated character string that contains the wrapper name.

Syntax

```
sqluint8* get_name()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the character string.

get_corelib function

Where defined

Wrapper

Purpose

Return a pointer to the null-terminated character string that contains the wrapper library name. This name is the wrapper library that is used in the CREATE WRAPPER statement. This name does not correspond to the library that is used during statement processing.

Syntax

```
sqluint8* get_corelib()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the character string.

get_version function

Where defined

Wrapper

Purpose

Return the version of the wrapper code.

Syntax

```
sqlint32 get_version()
```


Input arguments

None.

Output arguments

None.

Return value

Wrapper version.

get_info function**Where defined**

Wrapper

Purpose

Return a pointer to the wrapper catalog information object.

Syntax

Wrapper_Info* get_info()

Input arguments

None.

Output arguments

None.

Return value

Pointer to the catalog information object.

initialize_my_wrapper function**Where defined**

Wrapper

Purpose

Initialize the wrapper object state from the catalog information object. The default version does nothing.

Usage This member function can be implemented in the wrapper-specific subclass of Unfenced_Generic_Wrapper if wrapper-specific wrapper options are supported.

Syntax

virtual sqlint32 initialize_my_wrapper (Wrapper_Info* a_wrapper_info)

Unfenced_Generic_Wrapper

Input arguments

Table 120. Input arguments for the `initialize_my_wrapper` member function

Name	Data type	Description
<code>a_wrapper_info</code>	<code>Wrapper_Info*</code>	Catalog information object.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

`create_server` function

Where defined

Wrapper

Purpose

Instantiate the appropriate subclass of `Server` for the wrapper.

Usage This member function must be implemented by the wrapper in the wrapper-specific subclass of `Unfenced_Generic_Wrapper`.

Syntax

```
virtual Server* create_server (sqluint8* a_server_name,  
                              sqlint32* a_rc)
```

Input arguments

Table 121. Input arguments for the `create_server` member function

Name	Data type	Description
<code>a_server_name</code>	<code>sqluint8*</code>	Null-terminated name of the server.

Output arguments

Table 122. Output arguments for the `create_server` member function

Name	Data type	Description
<code>a_rc</code>	<code>sqlint32*</code>	Pointer to return code; 0 indicates success.

Return value

Pointer to the newly created server object or null.

Associated functions

The UnfencedWrapper_Hook function is not used by the Unfenced_Generic_Wrapper class, but is directly associated with this class.

UnfencedWrapper_Hook function**Purpose**

Hook function that enables the DB2 federated server to instantiate the wrapper-specific subclass of the Unfenced_Generic_Wrapper. This function must be implemented by the wrapper and must be exported from the unfenced wrapper module.

Syntax

```
extern "C" UnfencedWrapper* UnfencedWrapper_Hook()
```

Input arguments

None.

Output arguments

None.

Return value

Instantiated subclass of Unfenced_Generic_Wrapper or null if an error occurs.

Related reference:

- “Wrapper classes for the C++ API” on page 67

Fenced_Generic_Wrapper class

This topic describes the Fenced_Generic_Wrapper class and provides details for the constructor, the destructor, and the member functions.

Overview

The Fenced_Generic_Wrapper class represents the wrapper on the fenced process space.

The Fenced_Generic_Wrapper class is one of the wrapper classes for the C++ API.

Usage The wrapper must implement a subclass of Fenced_Generic_Wrapper. This class is instantiated by the wrapper-specific FencedWrapper_Hook function.

File sqlqg_fenced_generic_wrapper.h

Data Members

Fenced_Generic_Wrapper

The following table lists the data members that you can use with the Fenced_Generic_Wrapper class.

Table 123. Data members for the Fenced_Generic_Wrapper class

Name	Data type	Description
m_info	Wrapper_Info*	Contains the catalog information for the wrapper.
m_name	sqluint8*	Pointer to the character string that contains the name of the wrapper.
m_corelib	sqluint8*	Pointer to the character string that contains the name of the wrapper library.
m_version	sqlint32	Version of the wrapper code.

Constructors, destructors, and member functions

The following tables describe the constructor, the destructor, and the member functions of the Fenced_Generic_Wrapper class. The constructor, destructor, and functions are described in more detail after the tables.

Table 124. Constructors for the Fenced_Generic_Wrapper class

Constructor	Description
FencedGeneric_Wrapper	Construct an instance of the Fenced_Generic_Wrapper class.

Table 125. Destructors for the Fenced_Generic_Wrapper class

Destructor	Description
~FencedGeneric_Wrapper	Destructor for the Fenced_Generic_Wrapper class.

Table 126. Member functions for the Fenced_Generic_Wrapper class

Member function	Description
get_name	Return a pointer to the null-terminated character string that contains the wrapper name.
get_corelib	Return a pointer to the null-terminated character string that contains the wrapper library name.
get_version	Return the version of the wrapper code.

Table 126. Member functions for the Fenced_Generic_Wrapper class (continued)

Member function	Description
get_info	Return a pointer to the wrapper catalog information object.
initialize_my_wrapper	Initialize the wrapper object state from the catalog information object.
create_server	Instantiate the appropriate subclass of Server for the wrapper.

FencedGeneric_Wrapper constructor

Purpose

Construct an instance of the Fenced_Generic_Wrapper class.

Syntax

```
FencedGeneric_Wrapper (sqlint32* a_rc,
                       sqlint32 a_wrapper_version = 0)
```

Input arguments

Table 127. Input arguments for the FencedGeneric_Wrapper constructor

Name	Data type	Description
a_wrapper_version	sqlint32	Version of the wrapper code.

Output arguments

Table 128. Output arguments for the FencedGeneric_Wrapper constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code. This value must be 0 for success.

Return value

None.

~FencedGeneric_Wrapper destructor

Purpose

Destructor for the Fenced_Generic_Wrapper class.

Syntax

```
~FencedGeneric_Wrapper ()
```

Fenced_Generic_Wrapper

Input arguments

None.

Output arguments

None.

Return value

None.

get_name function

Where defined

Wrapper

Purpose

Return a pointer to the null-terminated character string that contains the wrapper name.

Syntax

```
sqluint8* get_name()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the character string.

get_corelib function

Where defined

Wrapper

Purpose

Return a pointer to the null-terminated character string that contains the wrapper library name.

Syntax

```
sqluint8* get_corelib()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the character string.

get_version function

Where defined
Wrapper

Purpose
Return the version of the wrapper code.

Syntax
`sqlint32 get_version()`

Input arguments
None.

Output arguments
None.

Return value
Wrapper version.

get_info function

Where defined
Wrapper

Purpose
Return a pointer to the wrapper catalog information object.

Syntax
`Wrapper_Info* get_info()`

Input arguments
None.

Output arguments
None.

Return value
Pointer to the catalog information object.

initialize_my_wrapper function

Where defined
Wrapper

Purpose
Initialize the wrapper object state from the catalog information object.
The default version does nothing.

Fenced_Generic_Wrapper

Usage The wrapper can implement the wrapper-specific subclass of `Fenced_Generic_Wrapper` if wrapper-specific wrapper options are supported.

Syntax

```
virtual sqlint32 initialize_my_wrapper (Wrapper_Info* a_wrapper_info)
```

Input arguments

Table 129. Input arguments for the initialize_my_wrapper member function

Name	Data type	Description
a_wrapper_info	Wrapper_Info*	Catalog information object.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

create_server function

Where defined

Wrapper

Purpose

Instantiate the appropriate subclass of `Server` for the wrapper.

Usage The wrapper must override this member function in the wrapper-specific subclass of `Fenced_Generic_Wrapper`

Syntax

```
virtual Server* create_server (sqluint8* a_server_name,  
                              sqlint32* a_rc)
```

Input arguments

Table 130. Input arguments for the create_server member function

Name	Data type	Description
a_server_name	sqluint8*	Null-terminated name of the server.

Output arguments

Table 131. Output arguments for the create_server member function

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code; 0 indicates success.

Return value

Pointer to the newly created server object or null.

Associated functions

The FencedWrapper_Hook function is not used by the Fenced_Generic_Wrapper class, but is directly associated with this class.

FencedWrapper_Hook function**Purpose**

Hook function that enables the DB2 federated server to instantiate the wrapper-specific subclass of the Fenced_Generic_Wrapper. This function must be implemented by the wrapper and must be exported from the fenced wrapper module.

Syntax

```
extern "C" FencedWrapper* FencedWrapper_Hook()
```

Input arguments

None.

Output arguments

None.

Return value

Instantiated subclass of Fenced_Generic_Wrapper or null if an error occurs.

Related reference:

- “Wrapper classes for the C++ API” on page 67

Server classes for the C++ API

The following table describes each server class for the C++ API.

Table 132. Server classes

Class name	Description
Unfenced_Generic_Server	The Unfenced_Generic_Server class is a subclass of the Unfenced_Server class and is the abstract base class for all of the server functionality that operates in the unfenced (trusted) process space. The Unfenced_Generic_Server class is responsible for validating the catalog information from the CREATE SERVER and ALTER SERVER statements.

Fenced_Generic_Wrapper

Table 132. Server classes (continued)

Class name	Description
Fenced_Generic_Server	The Fenced_Generic_Server class is a subclass of the Server class and is the abstract base class for all server functionality that operates in the fenced (untrusted) process space. The Fenced_Generic_Server class is responsible for creating remote connections and nicknames.

Related reference:

- “Unfenced_Generic_Server class” on page 82
- “Fenced_Generic_Server class” on page 93

Unfenced_Generic_Server class

This topic describes the Unfenced_Generic_Server class and provides details for the constructor and the member functions.

Overview

The Unfenced_Generic_Server class is a subclass of the Unfenced_Server class and is the abstract base class for all of the server functionality that operates in the unfenced (trusted) process space. This class is responsible for validating the catalog information from the CREATE SERVER and ALTER SERVER statements.

The Unfenced_Generic_Server class is one of the server classes for the C++ API.

Usage The wrapper must implement a subclass of Unfenced_Generic_Server. The Unfenced_Generic_Server class is instantiated by the wrapper in the create_server() method of the wrapper-specific subclass of the Unfenced_Generic_Wrapper class.

File sqlqg_unfenced_generic_server.h

Data members

The following table lists the data members that you can use with the Unfenced_Generic_Server class.

Table 133. Data members for the Unfenced_Generic_Server class

Name	Data type	Description
info	Server_Info*	Catalog information for the server.
name	sqluint8*	Null-terminated character string that contains the server name.

Table 133. Data members for the Unfenced_Generic_Server class (continued)

Name	Data type	Description
kind	server_kind	Kind of server which must be Server::generic_kind.
wrapper	Wrapper*	Pointer to the associated wrapper object that owns this server.

Constructors and member functions

The following tables describe the constructor and the member functions of the Unfenced_Generic_Server class. The constructor and functions are described in more detail after the tables.

Table 134. Constructors for the Unfenced_Generic_Server class

Constructor	Description
Unfenced_Generic_server	Construct an instance of the Unfenced_Generic_Server class.

Table 135. Member functions for the Unfenced_Generic_Server class

Member function	Description
create_remote_user	Instantiate an appropriate subclass of Remote_User.
plan_request	Analyze a proposed plan and determine what portion, if any, can be pushed down to the data source.
get_selectivity	Calculate the selectivity of a list of predicates.
create_reply	Instantiate a reply object.
get_name	Return a pointer to the null-terminated server name.
get_type	Retrieve the null-terminated server type as specified on the CREATE SERVER statement from the catalog information.
get_version	Retrieve the null-terminated server version as specified on CREATE SERVER statement.
get_info	Return a pointer to the stored catalog information object.
initialize_my_server	Initialize the server instances from valid catalog information.
create_nickname	Instantiate an appropriate subclass of a nickname for this server.

Unfenced_Generic_Server

Table 135. Member functions for the Unfenced_Generic_Server class (continued)

Member function	Description
is_reserved_server_option	Identify an option that is supported by DB2 but is ignored by the wrapper.
verify_my_register_server_info	Validate information that is provided on the CREATE SERVER statement.
verify_my_alter_server_info	Validate information that is provided on the ALTER SERVER statement.

Unfenced_Generic_server constructor

Purpose

Construct an instance of the Unfenced_Generic_Server class.

Syntax

```
Unfenced_Generic_server (sqluint8*      a_server_name,  
                        UnfencedWrapper* a_wrapper,  
                        sqlint32*      a_rc)
```

Input arguments

Table 136. Input arguments for the Unfenced_Generic_server constructor

Name	Data type	Description
a_server_name	sqluint8*	Null-terminated string that contains the name of the server.
a_wrapper	UnfencedWrapper*	Pointer to the wrapper object that owns this server object.

Output arguments

Table 137. Output arguments for the Unfenced_Generic_server constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to return code; 0 indicates success.

Return value

None.

create_remote_user function

Purpose

Instantiate an appropriate subclass of Remote_User.

Usage This member function can be implemented by the wrapper in the

wrapper-specific unfenced server subclass. This member function must be implemented if a wrapper-specific subclass of the Unfenced_Generic_User class is implemented.

Syntax

```
virtual Remote_User* create_remote_user (sqluint8* a_user_name,
                                         sqlint32* a_rc)
```

Input arguments

Table 138. Input arguments for the create_remote_user member function

Name	Data type	Description
a_user_name	sqluint8*	Null-terminated string that contains the name of the user.

Output arguments

Table 139. Output arguments for the create_remote_user member function

Name	Data type	Description
a_rc	sqlint32*	Pointer to return code; 0 indicates success.

Return value

Pointer to the Remote_User object.

plan_request function

Purpose

Analyze a proposed plan and determine what portion, if any, can be pushed down to the data source.

Usage This member function must be implemented by the wrapper in the wrapper-specific unfenced server subclass.

Syntax

```
virtual sqlint32 plan_request (Request* a_req,
                              Reply** a_rpl) = 0
```

Input arguments

Table 140. Input arguments for the plan_request member function

Name	Data type	Description
a_req	Request*	Pointer to the request object that describes the proposed plan.

Output arguments

Table 141. Output arguments for the `plan_request` member function

Name	Data type	Description
<code>a_rpl</code>	Reply**	Pointer to the chain of reply objects that are constructed by the <code>plan_request</code> .

Return value

Return code. A value of 0 indicates success.

`get_selectivity` function

Purpose

Calculate the selectivity of a list of predicates.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced server subclass. The default version uses built-in DB2 Information Integrator formulas for calculating the selectivity. The wrapper uses these formulas and the known dependencies between predicate expressions.

Syntax

```
virtual sqlint32 get_selectivity (Predicate_List* a_pl,  
                                float*          a_selectivity)
```

Input arguments

Table 142. Input arguments for the `get_selectivity` member function

Name	Data type	Description
<code>a_pl</code>	Predicate_List*	List of predicates.

Output arguments

Table 143. Output arguments for the `get_selectivity` member function

Name	Data type	Description
<code>a_selectivity</code>	float*	Selectivity that is expressed as a value from 0.0 to 1.0.

Return value

Return code. A value of 0 indicates success.

create_reply function**Purpose**

Instantiate a reply object.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced server subclass and must be implemented if using a wrapper-specific Reply subclass. This method is overridden if the wrapper implements its own cost model by subclassing the reply.

Syntax

```
virtual sqlint32 create_reply (Request* a_req,
                             Reply** a_rpl)
```

Input arguments

Table 144. Input arguments for the create_reply member function

Name	Data type	Description
a_req	Request*	Pointer to a request object.

Output arguments

Table 145. Output arguments for the create_reply member function

Name	Data type	Description
a_rpl	Reply**	Pointer to the reply object.

Return value

Return code. A value of 0 indicates success.

get_name function**Where defined**

Server

Purpose

Return a pointer to the null-terminated server name.

Syntax

```
sqluint8* get_name()
```

Input arguments

None.

Output arguments

None.

Unfenced_Generic_Server

Return value

Pointer to the server name.

get_type function

Where defined

Server

Purpose

Retrieve the null-terminated server type as specified on the CREATE SERVER statement from the catalog information.

Usage This method is valid only *after* the server object is initialized. If the server type is not specified in the catalog, this method logs an error. However, no user error is generated.

Syntax

```
sqluint8* get_type()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server type.

get_version function

Where defined

Server

Purpose

Retrieve the null-terminated server version as specified on CREATE SERVER statement.

Usage This method is valid only after the server object is initialized. If the server type is not specified in the catalog, this method logs an error. However, no user error is generated.

Syntax

```
sqluint8* get_version()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server version.

get_info function**Where defined**

Server

Purpose

Return a pointer to the stored catalog information object.

Usage This method is valid only after server initialization.

Syntax

```
Server_Info* get_info()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the catalog information object.

initialize_my_server function**Where defined**

Server

Purpose

Initialize the server instances from valid catalog information.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced server subclass and must be implemented if wrapper-specific server options are supported.

Syntax

```
virtual sqlint32 initialize_my_server (Server_Info* a_server_info)
```

Input arguments

Table 146. Input arguments for the initialize_my_server member function

Name	Data type	Description
a_server_info	Server_Info*	Validated catalog information.

Unfenced_Generic_Server

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

create_nickname function

Where defined

Server

Usage This member function must be implemented by the wrapper in the wrapper-specific unfenced server subclass.

Purpose

Instantiate an appropriate subclass of a nickname for this server.

Syntax

```
virtual Nickname* create_nickname (sqluint8* a_schema_name,  
                                   sqluint8* a_nickname_name,  
                                   sqlint32* a_rc)
```

Input arguments

Table 147. Input arguments for the create_nickname member function

Name	Data type	Description
a_schema_name	sqluint8*	Null-terminated string that contains the local (DB2) schema name of the nickname.
a_nickname_name	sqluint8*	Null-terminated string that contains the local (DB2) nickname name.

Output arguments

Table 148. Output arguments for the create_nickname member function

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code; a value of 0 indicates success.

Return value

Pointer to the instantiated nickname object.

is_reserved_server_option function**Where defined**

UnfencedServer

Purpose

Identify an option that is supported by DB2 but is ignored by the wrapper.

Syntax

```
virtual sqlint32 is_reserved_server_option (sqluint8* a_op_name)
```

Input arguments

Table 149. Input arguments for the is_reserved_server_option member function

Name	Data type	Description
a_op_name	sqluint8*	Null-terminated string that contains an option name.

Output arguments

None.

Return value

A value of 0 indicates that the option is not a DB2-defined server option. A non-zero value indicates that the option is a DB2-defined server option.

verify_my_register_server_info function**Where defined**

UnfencedServer

Purpose

Validate information that is provided on the CREATE SERVER statement.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced server subclass and must be implemented if wrapper-specific server options are supported. The wrapper must check whether an a_delta_info object has already been allocated before allocating its own.

Syntax

```
virtual sqlint32 verify_my_register_server_info
(Server_Info* a_server_info,
 Server_Info** a_delta_info)
```

Unfenced_Generic_Server

Input arguments

Table 150. Input arguments for the `verify_my_register_server_info` member function

Name	Data type	Description
<code>a_server_info</code>	<code>Server_Info*</code>	Information from the <code>CREATE SERVER</code> statement.

Output arguments

Table 151. Output arguments for the `verify_my_register_server_info` member function

Name	Data type	Description
<code>a_delta_info</code>	<code>Server_Info**</code>	Additional information that is provided by the wrapper.

Return value

Return code. A value of 0 indicates success.

`verify_my_alter_server_info` function

Where defined

`UnfencedServer`

Purpose

Validate information that is provided on the `ALTER SERVER` statement.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced server subclass and must be implemented if wrapper-specific server options are supported. The wrapper must check whether an `a_delta_info` object is already allocated before allocating its own.

Syntax

```
virtual sqlint32 verify_my_alter_server_info
    (Server_Info* a_server_info,
     Server_Info** a_delta_info)
```

Input arguments

Table 152. Input arguments for the `verify_my_alter_server_info` member function

Name	Data type	Description
<code>a_server_info</code>	<code>Server_Info*</code>	Information from an <code>ALTER SERVER</code> statement.

Output arguments*Table 153. Output arguments for the verify_my_alter_server_info member function*

Name	Data type	Description
a_delta_info	Server_Info**	Additional information that is provided by the wrapper.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Server classes for the C++ API” on page 81

Fenced_Generic_Server class

This topic describes the Fenced_Generic_Server class and provides details for the constructor and the member functions.

Overview

The Fenced_Generic_Server class is a subclass of the Server class and is the abstract base class for all server functionality that operates in the fenced (untrusted) process space. This class is responsible for creating remote connections and nicknames.

The Fenced_Generic_Server class is one of the server classes for the C++ API.

Usage The wrapper must implement a subclass of Fenced_Generic_Server. This class is instantiated by the wrapper in the create_server() method of the wrapper-specific subclass of Fenced_Generic_Wrapper.

File sqlqg_fenced_genserver.h

Data members

The following table lists the data members that you can use with the Fenced_Generic_Server class.

Table 154. Data members for the Fenced_Generic_Server class

Name	Data type	Description
info	Server_Info*	Catalog information for the server.
name	sqluint8*	Null-terminated character string that contains the server name.
kind	server_kind	Kind of server which must be Server::generic_kind.

Fenced_Generic_Server

Table 154. Data members for the Fenced_Generic_Server class (continued)

Name	Data type	Description
wrapper	Wrapper*	Pointer to the associated wrapper object that owns this server.

Constructors and member functions

The following tables describe the constructor and the member functions of the Fenced_Generic_Server class. The constructor and functions are described in more detail after the tables.

Table 155. Constructors for the Fenced_Generic_Server class

Constructor	Description
Fenced_Generic_Server	Constructor for this class.

Table 156. Member functions for the Fenced_Generic_Server class

Member function	Description
create_remote_connection	Instantiate an appropriate subclass of Remote_Connection.
get_name	Return a pointer to the null-terminated server name.
get_type	Retrieve the null-terminated server type that is specified on the CREATE SERVER statement from the catalog information.
get_version	Retrieve the null-terminated server version that is specified on the CREATE SERVER statement.
get_info	Return a pointer to the stored catalog information object.
initialize_my_server	Initialize the server instances from valid catalog information.
create_remote_user	Instantiate an appropriate subclass of Remote_User for this wrapper.
create_nickname	Instantiate an appropriate subclass of Nickname for this server.

Fenced_Generic_Server constructor

Purpose

Constructor for this class.

Syntax

```
Fenced_Generic_Server (sqluint8* a_server_name,
                      FencedWrapper* wrapper,
                      sqlint32* rc)
```

Input arguments

Table 157. Input arguments for the *Fenced_Generic_Server* constructor

Name	Data type	Description
a_server_name	sqluint8*	Name of the server.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

create_remote_connection function

Where defined

FencedServer

Purpose

Instantiate an appropriate subclass of Remote_Connection.

Usage This member function must be implemented by the wrapper in the wrapper-specific fenced server subclass.

Syntax

```
virtual sqlint32 create_remote_connection
(FencedRemote_User* a_user,
 Remote_connection** a_connection)
```

Input arguments

Table 158. Input arguments for the *create_remote_connection* member function

Name	Data type	Description
a_user	FencedRemote_User*	Pointer to the object that represents the user that is used for the connection.

Output arguments

Table 159. Output arguments for the *create_remote_connection* member function

Name	Data type	Description
a_connection	Remote_Connection**	Pointer to the remote connection object.

Fenced_Generic_Server

Return value

Return code. A value of 0 indicates success.

get_name function

Where defined

Server

Purpose

Return a pointer to the null-terminated server name.

Syntax

```
sqluint8* get_name()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server name.

get_type function

Where defined

Server

Purpose

Retrieve the null-terminated server type that is specified on the CREATE SERVER statement from the catalog information.

Usage This method is valid only after the server object initializes (that is, when the initialize_server runs). If the server type is not specified in the catalog, calling this method logs an error. However, no user error is generated.

Syntax

```
sqluint8* get_type()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server type.

get_version function**Where defined**

Server

Purpose

Retrieve the null-terminated server version that is specified on the CREATE SERVER statement.

Usage This method is valid only after the server object initializes (that is, when the initialize_server runs). If the server type is not specified in the catalog, calling this method logs an error. However, no user error is generated.

Syntax

```
sqluint8* get_version()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server version.

get_info function**Where defined**

Server

Purpose

Return a pointer to the stored catalog information object.

Usage This method is valid only after server initialization.

Syntax

```
Server_Info* get_info()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the catalog information object.

Fenced_Generic_Server

initialize_my_server function

Where defined

Server

Purpose

Initialize the server instances from valid catalog information.

Usage This member function can be implemented by the wrapper in the wrapper-specific fenced server subclass. This member function must be implemented if wrapper-specific server options are supported.

Syntax

```
virtual sqlint32 initialize_my_server (Server_Info* a_server_info)
```

Input arguments

Table 160. Input arguments for the initialize_my_server member function

Name	Data type	Description
a_server_info	Server_Info*	Validated catalog information.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

create_remote_user function

Where defined

Server

Purpose

Instantiate an appropriate subclass of Remote_User for this wrapper.

Usage This member function can be implemented by the wrapper in the wrapper-specific fenced server subclass. This member function must be implemented if a wrapper-specific Fenced_Generic_User subclass is used.

Syntax

```
virtual Remote_User* create_remote_user (sqluint8* a_user_name,  
                                         sqlint32* a_rc)
```

Input arguments*Table 161. Input arguments for the create_remote_user member function*

Name	Data type	Description
a_user_name	sqluint8*	Name of the remote user.

Output arguments*Table 162. Output arguments for the create_remote_user member function*

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code; 0 indicates success.

Return value

Pointer to the instantiated Remote_User.

create_nickname function**Where defined**

Server

Purpose

Instantiate an appropriate subclass of Nickname for this server.

Usage This member function must be implemented by the wrapper in the wrapper-specific fenced server subclass.

Syntax

```
virtual Nickname* create_nickname (sqluint8* a_schema_name,
                                   sqluint8* a_nickname_name,
                                   sqlint32* a_rc)
```

Input arguments*Table 163. Input arguments for the create_nickname member function*

Name	Data type	Description
a_schema_name	sqluint8*	Null-terminated string that contains the local (DB2) schema name of the nickname.
a_nickname_name	sqluint8*	Null-terminated string that contains the local (DB2) nickname name.

Fenced_Generic_Server

Output arguments

Table 164. Output arguments for the `create_nickname` member function

Name	Data type	Description
<code>a_rc</code>	<code>sqlint32*</code>	Pointer to the return code; a value of 0 indicates success.

Return value

Pointer to the instantiated Nickname object.

Related reference:

- “Server classes for the C++ API” on page 81

User classes for the C++ API

The following table describes each user class for the C++ API.

Table 165. User classes

Class name	Description
<code>Unfenced_Generic_User</code>	This class represents a user mapping in the unfenced (trusted) process space. This class is responsible for validating information from the CREATE USER MAPPING and ALTER USER MAPPING statements.
<code>Fenced_Generic_User</code>	This class represents a user mapping in the fenced (untrusted) process space.

Related reference:

- “Unfenced_Generic_User class” on page 100
- “Fenced_Generic_User class” on page 106

Unfenced_Generic_User class

This topic describes the `Unfenced_Generic_User` class and provides details for the constructor, the destructor, and the member functions.

Overview

The `Unfenced_Generic_User` class represents a user mapping in the unfenced (trusted) process space. This class is responsible for validating information from the CREATE USER MAPPING and ALTER USER MAPPING statements.

The `Unfenced_Generic_User` class is one of the user classes for the C++ API.

Usage The wrapper must implement a subclass of `Unfenced_Generic_User` if

wrapper-specific options for the CREATE USER MAPPING or the ALTER USER MAPPING statement are used. This class is used only for option validation and is instantiated by the wrapper in the create_remote_user() method of the wrapper-specific subclass of Unfenced_Generic_Server.

File sqlqg_unfenced_generic_user.h

Data members

The following table lists the data members that you can use with the Unfenced_Generic_User class.

Table 166. Data members for the Unfenced_Generic_User class

Name	Data type	Description
local_name	sqluint8*	Null-terminated (local) user name.
server	Server*	Pointer to the Server object that owns this user.
info	User_Info*	Local copy of the catalog information. This data member is valid only after initialize_my_user.

Constructors, destructors, and member functions

The following tables describe the constructor, the destructor, and the member functions of the Unfenced_Generic_User class. The constructor, destructor, and functions are described in more detail after the tables.

Table 167. Constructors for the Unfenced_Generic_User class

Constructor	Description
Unfenced_Generic_User	Construct an instance of Unfenced_Generic_User.

Table 168. Destructors for the Unfenced_Generic_User class

Destructor	Description
~Unfenced_Generic_User	Destroy an instance of Unfenced_Generic_User.

Table 169. Member functions for the Unfenced_Generic_User class

Member function	Description
get_info	Return a pointer to the local copy of User_Info.

Unfenced_Generic_User

Table 169. Member functions for the Unfenced_Generic_User class (continued)

Member function	Description
is_reserved_user_option	Verify that a specified catalog option is a DB2-reserved option for a user mapping.
initialize_my_user	Initialize the state of the object from the valid catalog information.
verify_my_register_user_info	Validate information from the CREATE USER MAPPING statement.
verify_my_alter_user_info	Validate information from the ALTER USER MAPPING statement.

Unfenced_Generic_User constructor

Purpose

Construct an instance of Unfenced_Generic_User.

Syntax

```
Unfenced_Generic_User (sqluint8*    a_user_name,  
                       UnfencedServer* a_server,  
                       sqlint32*     a_rc)
```

Input arguments

Table 170. Input arguments for the Unfenced_Generic_User constructor

Name	Data type	Description
a_user_name	sqluint8*	Null-terminated (local) user name.
a_server	UnfencedServer*	Pointer to the Server object that owns this user.

Output arguments

Table 171. Output arguments for the Unfenced_Generic_User constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to the Server object that owns this user.

Return value

None.

~Unfenced_Generic_User destructor**Purpose**

Destroy an instance of Unfenced_Generic_User.

Syntax

```
virtual ~Unfenced_Generic_User ()
```

Input arguments

None.

Output arguments

None.

Return value

None.

get_info function**Purpose**

Return a pointer to the local copy of User_Info.

Syntax

```
User_Info* get_info ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the User_Info object.

is_reserved_user_option function**Purpose**

Verify that a specified catalog option is a DB2 reserved option for a user mapping.

Syntax

```
virtual sqluint32 is_reserved_user_option (sqluint8* a_opt_name)
```

Input arguments

Table 172. Input arguments for the is_reserved_user_option member function

Name	Data type	Description
a_opt_name	sqluint8*	Null-terminated option name to be checked.

Unfenced_Generic_User

Output arguments

None.

Return value

A value of 0 indicates that the option is not a DB2-reserved option.

initialize_my_user function

Purpose

Initialize the state of the object from the valid catalog information.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced user subclass.

Syntax

```
virtual sqlint32 initialize_my_user (User_Info* a_user_info)
```

Input arguments

Table 173. Input arguments for the initialize_my_user member function

Name	Data type	Description
a_user_info	User_Info*	Validated catalog information.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

verify_my_register_user_info function

Purpose

Validate information from the CREATE USER MAPPING statement.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced user subclass. This member function must be implemented if wrapper-specific user mapping options are supported.

The wrapper verifies whether an a_delta_info object was allocated before allocating one itself.

Syntax

```
virtual sqlint32 verify_my_register_user_info (User_Info* a_user_info,  
                                              User_Info** a_delta_info)
```


Input arguments

Table 174. Input arguments for the `verify_my_register_user_info` member function

Name	Data type	Description
<code>a_user_info</code>	<code>User_Info*</code>	Information from a CREATE USER MAPPING statement.

Output arguments

Table 175. Output arguments for the `verify_my_register_user_info` member function

Name	Data type	Description
<code>a_delta_info</code>	<code>User_Info**</code>	Additional information that is provided by the wrapper.

Return value

Return code. A value of 0 indicates success.

`verify_my_alter_user_info` function

Purpose

Validate information from the ALTER USER MAPPING statement.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced user subclass. This member function must be implemented if wrapper-specific user mapping options are supported.

The wrapper verifies whether an `a_delta_info` object was allocated before allocating one itself.

Syntax

```
virtual sqlint32 verify_my_alter_user_info (User_Info* a_user_info,
                                           User_Info** a_delta_info)
```

Input arguments

Table 176. Input arguments for the `verify_my_alter_user_info` member function

Name	Data type	Description
<code>a_user_info</code>	<code>User_Info*</code>	Information from the ALTER USER MAPPING statement.

Unfenced_Generic_User

Output arguments

Table 177. Output arguments for the `verify_my_alter_user_info` member function

Name	Data type	Description
<code>a_delta_info</code>	<code>User_Info**</code>	Additional information that is provided by the wrapper.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “User classes for the C++ API” on page 100

Fenced_Generic_User class

This topic describes the `Fenced_Generic_User` class and provides details for the constructor, the destructor, and the member functions.

Overview

The `Fenced_Generic_User` class represents a user mapping in the fenced (untrusted) process space.

The `Fenced_Generic_User` class is one of the user classes for the C++ API.

Usage This class is instantiated by the wrapper in the `create_remote_user()` method of the wrapper-specific subclass of `Fenced_Generic_Server`. The wrapper implements a subclass of `Fenced_Generic_User` if wrapper-specific user mapping options are supported.

File `sqlqg_fenced_generic_user.h`

Data members

The following table lists the data members that you can use with the `Fenced_Generic_User` class.

Table 178. Data members for the `Fenced_Generic_User` class

Name	Data type	Description
<code>local_name</code>	<code>sqluint8*</code>	Null-terminated (local) user name.
<code>server</code>	<code>Server*</code>	Pointer to the <code>Server</code> object that owns this user.
<code>info</code>	<code>User_Info*</code>	Local copy of the catalog information. This data member is valid only after <code>initialize_my_user</code> .

Constructors, destructors, and member functions

The following tables describe the constructor, the destructor, and the member functions of the Fenced_Generic_User class. The constructor, destructor, and functions are described in more detail after the tables.

Table 179. Constructors for the Fenced_Generic_User class

Constructor	Description
Fenced_Generic_User	Construct an instance of Fenced_Generic_User.

Table 180. Destructors for the Fenced_Generic_User class

Destructor	Description
~Fenced_Generic_User	Destroy an instance of Fenced_Generic_User.

Table 181. Member functions for the Fenced_Generic_User class

Member function	Description
get_info	Return a pointer to the local copy of User_Info.
initialize_my_user	Initialize the state of the object from the valid catalog information.

Fenced_Generic_User constructor

Purpose

Construct an instance of Fenced_Generic_User.

Syntax

```
Fenced_Generic_User (sqluint8*   a_local_user_name,
                    FencedServer* a_server,
                    sqlint32*    a_rc)
```

Input arguments

Table 182. Input arguments for the Fenced_Generic_User constructor

Name	Data type	Description
a_local_user_name	sqluint8*	Null-terminated (local) user name.
a_server	FencedServer*	Pointer to the Server object that owns this user.

Fenced_Generic_User

Output arguments

Table 183. Output arguments for the Fenced_Generic_User constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code; 0 indicates success.

Return value

None.

~Fenced_Generic_User destructor

Purpose

Destroy an instance of Fenced_Generic_User.

Syntax

```
virtual ~Fenced_Generic_User ()
```

Input arguments

None.

Output arguments

None.

Return value

None.

get_info function

Purpose

Return a pointer to the local copy of User_Info.

Syntax

```
User_Info* get_info ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the User_Info object.

initialize_my_user function**Purpose**

Initialize the state of the object from the valid catalog information.

Usage This member function can be implemented by the wrapper in the wrapper-specific fenced user subclass and must be implemented if wrapper-specific user mapping options are supported.

Syntax

```
virtual sqlint32 initialize_my_user (User_Info* a_user_info)
```

Input arguments

Table 184. Input arguments for the initialize_my_user member function

Name	Data type	Description
a_user_info	User_Info*	Validated catalog information.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “User classes for the C++ API” on page 100

Nickname classes for the C++ API

The following table describes each nickname class for the C++ API.

Table 185. Nickname classes

Class name	Description
Unfenced_Generic_Nickname	This class represents a nickname in the unfenced (trusted) process space. This class is responsible for validating information from CREATE NICKNAME and ALTER NICKNAME statements.
Fenced_Generic_Nickname	This class represents a nickname in the fenced (untrusted) process space. This class is responsible for validating information from the CREATE NICKNAME statement.

Related reference:

- “Unfenced_Generic_Nickname class” on page 110

- “Fenced_Generic_Nickname class” on page 119

Unfenced_Generic_Nickname class

This topic describes the Unfenced_Generic_Nickname class and provides details for the constructor and the member functions.

Overview

The Unfenced_Generic_Nickname class represents a nickname in the unfenced (trusted) process space. This class is responsible for validating information from CREATE NICKNAME and ALTER NICKNAME statements.

The Unfenced_Generic_Nickname class is one of the nickname classes for the C++ API.

Usage This class must be subclassed by the wrapper. This class is instantiated by the wrapper in the create_nickname() method of the wrapper-specific subclass of Unfenced_Generic_Server.

File sqlqg_unfenced_generic_nickname.h

Data members

The following table lists the data members that you can use with the Unfenced_Generic_Nickname class.

Table 186. Data members for the Unfenced_Generic_Nickname class

Name	Data type	Description
name	sqluint8*	Null-terminated string that contains the (local) name of the nickname.
schema	sqluint8*	Null-terminated string that contains the (local) name of the schema.
server	Server*	Pointer to the server object that owns this nickname.
m_cardinality	sqlint64	Cardinality.
m_advance_cost	sqlint32	Cost (time in milliseconds) to fetch a row.
m_setup_cost	sqlint32	Cost (time in milliseconds) to perform one-time setup of a query.
m_submission_cost	sqlint32	Cost (time in milliseconds) to submit a query (other than one-time setup costs.)

Constructors and member functions

The following tables describe the constructor and the member functions of the Unfenced_Generic_Nickname class. The constructor and functions are described in more detail after the tables.

Table 187. Constructors for the Unfenced_Generic_Nickname class

Constructor	Description
Unfenced_Generic_Nickname	Construct an instance of Unfenced_Generic_Nickname.

Table 188. Member functions for the Unfenced_Generic_Nickname class

Member function	Description
get_local_name	Return pointer to a null-terminated local nickname name.
get_local_schema	Return pointer to a null-terminated local schema name.
get_server	Return pointer to a containing server object.
is_reserved_nickname_option	Determine if a particular option name is one of the built-in DB2 nickname options.
is_reserved_column_option	Determine if a particular option name is one of the built-in DB2 column options.
initialize_my_nickname	Initialize the nickname with valid information from the catalog.
verify_my_register_nickname_info	Validate information from the CREATE NICKNAME statement.
verify_my_alter_nickname_info	Validate information from the ALTER NICKNAME statement.
get_card	Retrieve the cardinality for a nickname.
get_advance_cost	Retrieve the value of the ADVANCE_COST nickname option that is used in the default cost model.
get_setup_cost	Retrieve the value of the SETUP_COST nickname option that is used in the default cost model.
get_submission_cost	Retrieve the value of the SUBMISSION_COST nickname option that is used in the default cost model.

Unfenced_Generic_Nickname

Unfenced_Generic_Nickname constructor

Purpose

Construct an instance of Unfenced_Generic_Nickname.

Syntax

```
Unfenced_Generic_Nickname (sqluint8* a_schema,  
                           sqluint8*   a_nickname_name,  
                           Unfenced_Generic_Server* a_nickname_server,  
                           sqlint32*   a_rc)
```

Input arguments

Table 189. Input arguments for the Unfenced_Generic_Nickname constructor

Name	Data type	Description
a_schema	sqluint8*	Null-terminated schema name.
a_nickname_name	sqluint8*	Null-terminated (local) nickname name.
a_nickname_server	Unfenced_Generic_Server*	Pointer to a containing server object.

Output arguments

Table 190. Output arguments for the Unfenced_Generic_Nickname constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to return code; 0 indicates success.

Return value

None.

get_local_name function

Purpose

Return pointer to a null-terminated local nickname name.

Syntax

```
sqluint8* get_local_name ()
```

Input arguments

None.

Output arguments

None.

Return value

Null-terminated nickname name.

get_local_schema function**Purpose**

Return pointer to a null-terminated local schema name.

Syntax

```
sqluint8* get_local_schema ()
```

Input arguments

None.

Output arguments

None.

Return value

Null-terminated schema name.

get_server function**Purpose**

Return pointer to a containing server object.

Syntax

```
Server* get_server ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to a containing server object.

is_reserved_nickname_option function**Purpose**

Determine if a particular option name is one of the built-in DB2 nickname options.

Syntax

```
virtual sqluint32 is_reserved_nickname_option (sqluint8* a_opt_name)
```

Unfenced_Generic_Nickname

Input arguments

Table 191. Input arguments for the `is_reserved_nickname_option` member function

Name	Data type	Description
<code>a_opt_name</code>	<code>sqluint8*</code>	Null-terminated option name string.

Output arguments

None.

Return value

A value of 0 indicates that the option is not a reserved nickname option. A non-zero return value indicates that the option is a reserved nickname option.

`is_reserved_column_option` function

Purpose

Determine if a particular option name is one of the built-in DB2 column options.

Syntax

```
virtual sqluint32 is_reserved_column_option (sqluint8* a_opt_name)
```

Input arguments

Table 192. Input arguments for the `is_reserved_column_option` member function

Name	Data type	Description
<code>a_opt_name</code>	<code>sqluint8*</code>	Null-terminated option name string.

Output arguments

None.

Return value

A value of 0 indicates that the option is not a reserved option. A non-zero return value indicates that the option is a reserved column option.

`initialize_my_nickname` function

Purpose

Initialize the nickname with valid information from the catalog.

Usage This member function can be implemented by the wrapper in the wrapper-specific unfenced nickname subclass.

Syntax

```
virtual sqlint32 initialize_my_nickname (Nickname_Info* a_cat_info)
```

Input arguments

Table 193. Input arguments for the initialize_my_nickname member function

Name	Data type	Description
a_cat_info	Nickname_Info*	Information from the catalog.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

verify_my_register_nickname_info function**Purpose**

Validate information from the CREATE NICKNAME statement.

Usage This member function can be implemented by the wrapper in the wrapper-specific fenced nickname subclass. Either this method or the same method of the fenced class must be implemented if the wrapper-specific nickname or column options are supported. Because the verify_my_register_nickname_info function is part of the trusted process space, this member function cannot interact with the remote data source. If interaction is necessary to verify the nickname information, the verify_my_register_nickname_info member function of the Fenced_Generic_Nickname class must be implemented.

The wrapper checks whether an a_delta_info object was previously allocated before allocating one itself. This method is called after Fenced_Generic_Nickname::verify_my_register_nickname_info is called.

Syntax

```
virtual sqlint32 verify_my_register_nickname_info
(Nickname_Info* a_nick_info,
 Nickname_Info** a_delta_info)
```

Input arguments

Table 194. Input arguments for the `verify_my_register_nickname_info` member function

Name	Data type	Description
<code>a_nick_info</code>	<code>Nickname_Info*</code>	Information from the <code>CREATE NICKNAME</code> statement.

Output arguments

Table 195. Output arguments for the `verify_my_register_nickname_info` member function

Name	Data type	Description
<code>a_delta_info</code>	<code>Nickname_Info**</code>	Information that is added by the wrapper.

Return value

Return code. A value of 0 indicates success.

`verify_my_alter_nickname_info` function

Purpose

Validate information from the `ALTER NICKNAME` statement.

Usage This member function can be implemented by the wrapper in the wrapper-specific nickname subclass. This member function must be implemented if the wrapper-specific nickname or column options are supported. Because the `verify_my_alter_nickname_info` function is part of the trusted process space, this member function cannot interact with the remote data source.

The wrapper checks whether an `a_delta_info` object was previously allocated before allocating one itself.

Syntax

```
virtual sqlint32 verify_my_alter_nickname_info  
    (Nickname_Info* a_nick_info,  
     Nickname_Info** a_delta_info)
```

Input arguments

Table 196. Input arguments for the `verify_my_alter_nickname_info` member function

Name	Data type	Description
<code>a_nick_info</code>	<code>Nickname_Info*</code>	Information from the <code>ALTER NICKNAME</code> statement.

Output arguments*Table 197. Output arguments for the verify_my_alter_nickname_info member function*

Name	Data type	Description
a_delta_info	Nickname_Info**	Information that is added by the wrapper.

Return value

Return code. A value of 0 indicates success.

get_card function**Purpose**

Retrieve the cardinality for a nickname. The cardinality is stored in the DB2 Information Integrator system catalog.

Syntax

```
void get_card (sqlint64* a_cardinality) const
```

Input arguments

None.

Output arguments*Table 198. Output arguments for the get_card member function*

Name	Data type	Description
a_cardinality	sqlint64*	Cardinality.

Return value

None.

get_advance_cost function**Purpose**

Retrieve the value of the ADVANCE_COST nickname option that is used in the default cost model.

Syntax

```
sqlint32 get_advance_cost (void) const
```

Input arguments

None.

Unfenced_Generic_Nickname

Output arguments

None.

Return value

Value of the ADVANCE_COST nickname option or the default value if the option is not specified for the nickname.

get_setup_cost function

Purpose

Retrieve the value of the SETUP_COST nickname option that is used in the default cost model.

Syntax

```
sqlint32 get_setup_cost (void) const
```

Input arguments

None.

Output arguments

None.

Return value

Value of the SETUP_COST nickname option or the default value if the option is not specified for the nickname.

get_submission_cost function

Purpose

Retrieve the value of the SUBMISSION_COST nickname option that is used in the default cost model.

Syntax

```
sqlint32 get_submission_cost (void) const
```

Input arguments

None.

Output arguments

None.

Return value

Value of the SUBMISSION_COST nickname option or the default value if the option is not specified for the nickname.

Related reference:

- “Nickname classes for the C++ API” on page 109

Fenced_Generic_Nickname class

This topic describes the Fenced_Generic_Nickname class and provides details for the constructor and the member functions.

Overview

The Fenced_Generic_Nickname class represents a nickname in the fenced (untrusted) process space. This class is responsible for validating information from the CREATE NICKNAME statement.

The Fenced_Generic_Nickname class is one of the nickname classes for the C++ API.

Usage This class must be subclassed by the wrapper and is instantiated by the wrapper in the create_nickname() method of the wrapper-specific subclass of Fenced_Generic_Server.

File sqlqg_fenced_generic_nickname.h

Data members

The following table lists the data members that you can use with the Fenced_Generic_Nickname class.

Table 199. Data members for the Fenced_Generic_Nickname class

Name	Data type	Description
name	sqluint8*	Null-terminated string that contains the (local) name of the nickname.
schema	sqluint8*	Null-terminated string that contains the (local) name of the schema.
server	Server*	Pointer to the server object that owns this nickname.

Constructors and member functions

The following tables describe the constructor and the member functions of the Fenced_Generic_Nickname class. The constructor and functions are described in more detail after the tables.

Table 200. Constructors for the Fenced_Generic_Nickname class

Constructor	Description
Fenced_Generic_Nickname	Construct an instance of Fenced_Generic_Nickname.

Fenced_Generic_Nickname

Table 201. Member functions for the *Fenced_Generic_Nickname* class

Member function	Description
<code>get_local_name</code>	Return pointer to the null-terminated local nickname name.
<code>get_local_schema</code>	Return pointer to the null-terminated local schema name.
<code>get_server</code>	Return pointer to the containing server object.
<code>is_reserved_nickname_option</code>	Determine if a particular option name is one of the built-in DB2 nickname options.
<code>is_reserved_column_option</code>	Determine if a particular option name is one of the built-in DB2 column options.
<code>initialize_my_nickname</code>	Initialize the nickname with valid information from the catalog.
<code>verify_my_register_nickname_info</code>	Validate information from the CREATE NICKNAME statement.

Fenced_Generic_Nickname constructor

Purpose

Construct an instance of *Fenced_Generic_Nickname*.

Syntax

```
Fenced_Generic_Nickname (sqluint8*      a_schema,  
                        sqluint8*      a_name,  
                        Fenced_Generic_Server* a_server,  
                        sqlint32*      a_rc)
```

Input arguments

Table 202. Input arguments for the *Fenced_Generic_Nickname* constructor

Name	Data type	Description
<code>a_schema</code>	<code>sqluint8*</code>	Null-terminated schema name.
<code>a_name</code>	<code>sqluint8*</code>	Null-terminated (local) nickname name.
<code>a_server</code>	<code>Fenced_Generic_Server*</code>	Pointer to the containing server object.

Output arguments*Table 203. Output arguments for the Fenced_Generic_Nickname constructor*

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code; 0 indicates success.

Return value

None.

get_local_name function**Purpose**

Return pointer to the null-terminated local nickname name.

Syntax

sqluint8* get_local_name ()

Input arguments

None.

Output arguments

None.

Return value

Null-terminated nickname name.

get_local_schema function**Purpose**

Return pointer to the null-terminated local schema name.

Syntax

sqluint8* get_local_schema ()

Input arguments

None.

Output arguments

None.

Return value

Null-terminated schema name.

Fenced_Generic_Nickname

get_server function

Purpose

Return pointer to the containing server object.

Syntax

```
Server* get_server ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to containing server object.

is_reserved_nickname_option function

Purpose

Determine if a particular option name is one of the built-in DB2 nickname options.

Syntax

```
virtual sqluint32 is_reserved_nickname_option (sqluint8* a_opt_name)
```

Input arguments

Table 204. Input arguments for the is_reserved_nickname_option member function

Name	Data type	Description
a_opt_name	sqluint8*	Null-terminated option name string.

Output arguments

None.

Return value

A value of 0 indicates that the option is not a reserved option. A non-zero return value indicates that the option is a reserved nickname option.

is_reserved_column_option function

Purpose

Determine if a particular option name is one of the built-in DB2 column options.

Syntax

```
virtual sqluint32 is_reserved_column_option (sqluint8* a_opt_name)
```

Input arguments

Table 205. Input arguments for the `is_reserved_column_option` member function

Name	Data type	Description
<code>a_opt_name</code>	<code>sqluint8*</code>	Null-terminated option name string.

Output arguments

None.

Return value

A value of 0 indicates that the option is not a reserved option. A non-zero return value indicates that the option is a reserved column option.

initialize_my_nickname function

Purpose

Initialize the nickname with valid information from the catalog.

Usage The wrapper can implement this member function in the wrapper-specific fenced nickname subclass.

Syntax

```
virtual sqlint32 initialize_my_nickname (Nickname_Info* a_nick_info)
```

Input arguments

Table 206. Input arguments for the `initialize_my_nickname` member function

Name	Data type	Description
<code>a_nick_info</code>	<code>Nickname_Info*</code>	Information from the catalog.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

verify_my_register_nickname_info function

Purpose

Validate information from the CREATE NICKNAME statement.

Usage The wrapper can implement this member function in the

Fenced_Generic_Nickname

wrapper-specific fenced nickname subclass. Either this method or the same method of the unfenced class must be implemented if the wrapper-specific nickname or column options are supported. Implement this method if the wrapper must interact with the remote data source when verifying nickname information.

The wrapper checks whether an `a_delta_info` object was previously allocated before allocating one itself. This method is called before `Unfenced_Generic_Nickname::verify_my_register_nickname_info` is called.

Syntax

```
virtual sqlint32 verify_my_register_nickname_info  
    (Nickname_Info* a_nick_info,  
     Nickname_Info** a_delta_info)
```

Input arguments

Table 207. Input arguments for the `verify_my_register_nickname_info` member function

Name	Data type	Description
<code>a_nick_info</code>	<code>Nickname_Info*</code>	Information from the CREATE NICKNAME statement.

Output arguments

Table 208. Output arguments for the `verify_my_register_nickname_info` member function

Name	Data type	Description
<code>a_delta_info</code>	<code>Nickname_Info**</code>	Information that is added by the wrapper.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Nickname classes for the C++ API” on page 109

Remote_Connection class

This topic describes the `Remote_Connection` class and provides details for the constructor and the member functions.

Overview

The Remote_Connection class represents a connection between DB2 and the remote data source. This class manages the connection, creates the remote operation objects, and maintains the remote operation objects.

The Remote_Connection class is a connection class for the C++ API.

Usage The wrapper can subclass this class to create wrapper-specific connection subclasses. This class is instantiated by the create_remote_connection() method of the wrapper-specific Fenced_Generic_Server subclass.

File sqlqg_connection.h

Data members

The following table lists the data members that you can use with the Remote_Connection class.

Table 209. Data members for the Remote_Connection class

Name	Data type	Description
kind	connection_kind	Kind of connection (1 phase or 2 phase).
server	FencedServer*	Pointer to the server that owns this connection.
user	FencedRemote_User*	The user who is associated with the connection.
connect	unsigned short	The flag that indicates that the connection is active.

Constructors and member functions

The following tables describe the constructor and the member functions of the Remote_Connection class. The constructor and functions are described in more detail after the tables.

Table 210. Constructors for the Remote_Connection class

Constructor	Description
Remote_Connection	Construct a Remote_Connection object.

Table 211. Member functions for the Remote_Connection class

Member function	Description
connect	Establish a connection with the remote data source.

Remote_Connection

Table 211. Member functions for the Remote_Connection class (continued)

Member function	Description
disconnect	Destroy a connection with the remote data source.
is_connected	Indicate whether the connection is active or inactive.
create_remote_query	Create an appropriate subclass of Remote_Query.
create_remote_passthru	Create an appropriate subclass of Remote_Passthru.
get_server	Return a pointer to the containing server.
get_user	Return a pointer to the associated Remote_User object.
connect	Indicate the successful completion of a transaction and that the remote data source then commits the transaction.
rollback	Indicate the unsuccessful completion of a transaction and that the remote data source then rolls back the transaction.

Remote_Connection constructor

Purpose

Construct a Remote_Connection object.

Syntax

```
Remote_Connection (FencedServer*   remote_server,  
                  FencedRemote_User* remote_user,  
                  connection_kind   k=one_phase_kind,  
                  sqlint32*         rc = 0)
```

Input arguments

Table 212. Input arguments for the Remote_Connection constructor

Name	Data type	Description
FencedServer*	remote_server	Server that owns this connection.
remote_user	FencedRemote_User*	User that is associated with the connection.
k	connection_kind	Kind of connection.

Output arguments*Table 213. Output arguments for the Remote_Connection constructor*

Name	Data type	Description
rc	sqlint32*	Pointer to the return code; 0 indicates success.

Return value

None.

connect function**Purpose**

Establish a connection with the remote data source.

Usage DB2 Information Integrator invokes this member function to open a connection to the remote data source. This member function must be implemented by the wrapper in the wrapper-specific Remote_Connection subclass. The default behavior is to report an error.

Syntax

```
virtual sqlint32 connect ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

disconnect function**Purpose**

Destroy a connection with the remote data source.

Usage DB2 Information Integrator invokes this member function to close a connection to the remote data source. This member function must be implemented by the wrapper in the wrapper-specific Remote_Connection subclass. The default behavior reports an error.

Syntax

```
virtual sqlint32 disconnect ()
```

Remote_Connection

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

is_connected function

Purpose

Indicate whether the connection is active or inactive.

Syntax

```
unsigned short is_connected ()
```

Input arguments

None.

Output arguments

None.

Return value

A value of TRUE if the connection is active. Otherwise, the return value is FALSE.

create_remote_query function

Purpose

Create an appropriate subclass of Remote_Query.

Usage DB2 Information Integrator invokes this member function to instantiate the wrapper-specific Remote_Query subclass. The wrapper must implement this member function in the wrapper-specific Remote_Connection subclass.

Syntax

```
virtual sqlint32 create_remote_query  
                (Runtime_Operation* runtime_query,  
                 Remote_Query**    query)
```

Input arguments

Table 214. Input arguments for the create_remote_query member function

Name	Data type	Description
runtime_query	Runtime_Operation*	DB2 internal structure that describes the operation.

Table 214. Input arguments for the `create_remote_query` member function (continued)

Name	Data type	Description
<code>query</code>	<code>Remote_Query**</code>	Pointer to the allocated <code>Remote_Query</code> object.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

create_remote_passthru function**Purpose**Create an appropriate subclass of `Remote_Passthru`.

Usage DB2 Information Integrator invokes this member function to instantiate the wrapper-specific `Remote_Passthru` subclass. The wrapper can implement this member function in the wrapper-specific `Remote_Connection` subclass. This member function is necessary only if the wrapper supports the PASSTHRU mode.

Syntax

```
virtual sqlint32 create_remote_passthru
                (Runtime_Operation* runtime_passthru,
                 Remote_Passthru**  query)
```

Input argumentsTable 215. Input arguments for the `create_remote_passthru` member function

Name	Data type	Description
<code>runtime_passthru</code>	<code>Runtime_Operation*</code>	DB2 internal structure that describes the operation.
<code>query</code>	<code>Runtime_Passthru**</code>	Pointer to the allocated <code>Remote_Passthru</code> object.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Remote_Connection

get_server function

Purpose

Return a pointer to the containing server.

Syntax

```
FencedServer* get_server()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server.

get_user function

Purpose

Return a pointer to the associated Remote_User object.

Syntax

```
FencedRemote_User* get_user ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the remote user object.

commit function

Purpose

Indicate the successful completion of a transaction and that the remote data source then commits the transaction.

Usage DB2 Information Integrator invokes this member function to indicate the successful completion of a transaction. This member function must be implemented by the wrapper in the wrapper-specific Remote_Connection subclass. The default behavior is to report an error.

Syntax

```
sqlint32 commit ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

rollback function**Purpose**

Indicate the unsuccessful completion of a transaction and that the remote data source then rolls back the transaction.

Usage This member function is invoked by DB2 Information Integrator to indicate the unsuccessful completion of a transaction and that the remote data source rolls back the transaction. This member function can be implemented by the wrapper in the wrapper-specific Remote_Connection subclass. The default behavior is to report an error.

Syntax

```
sqlint32 rollback ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Server classes for the C++ API” on page 81

Operation classes for the C++ API

The following table describes each operation class for the C++ API.

Table 216. Operation classes

Class name	Description
Remote_Query	This class encapsulates the information that is necessary to run a query on the remote data source. The wrapper must subclass this class to implement the operation.

Remote_Connection

Table 216. Operation classes (continued)

Class name	Description
Remote_Passthru	This class represents a pass-through operation at a remote data source. A wrapper must subclass the Remote_Passthru class to support pass-through operations.

Related reference:

- “Remote_Query class” on page 132
- “Remote_Passthru class” on page 142

Remote_Query class

This topic describes the Remote_Query class and provides details for the constructor and the member functions.

Overview

The Remote_Query class encapsulates the information that is necessary to run a query on the remote data source. The wrapper must subclass this class to implement the operation.

The Remote_Query class is one of the operation classes for the C++ API.

Usage This class is instantiated by the wrapper in the create_remote_query() method of the wrapper-specific Remote_Connection subclass.

File sqlqg_operation.h

Parent class
Remote_Operation.

Data members
None.

Constructors and member functions

The following tables describe the constructor and the member functions of the Remote_Query class. The constructor and functions are described in more detail after the tables.

Table 217. Constructors for the Remote_Query class

Constructor	Description
Remote_Query	Construct a Remote_Query object.

Table 218. Member functions for the Remote_Query class

Member function	Description
get_connection	Retrieve a pointer to the connection that owns this operation.
get_server	Retrieve a pointer to the server that owns this operation.
get_input_data	Retrieve a pointer to the list of input values to the operation.
get_output_data	Retrieve a pointer to the list of output data buffers.
get_exec_desc	Retrieve a pointer and the length for the execution descriptor for this query.
open	Open a query.
reopen	Reopen a query.
fetch	Fetch a single row.
close	Close a query.
report_eof	Report an end-of-file condition during a fetch.
get_status	Retrieve the current query status.
set_status	Set the current query status.
fetch_lob	Retrieve LOB data into a special LOB buffer that is provided by DB2.
set_lob_next	Suspend the processing of non-LOB columns and start the processing of one or more LOB columns.
lob_data_ready	Indicates that the wrapper transferred a section of LOB data to the buffer.

Remote_Query constructor

Purpose

Construct a Remote_Query object.

Syntax

```
Remote_Query (Remote_Connection* a_active_connection,
              Runtime_Operation* a_runtime_info,
              sqlint32*          a_rc)
```

Remote_Query

Input arguments

Table 219. Input arguments for the Remote_Query constructor

Name	Data type	Description
a_active_connection	Remote_Connection*	Connection for this query.
a_runtime_info	Runtime_Operation*	Internal data for this query.

Output arguments

Table 220. Output arguments for the Remote_Query constructor

Name	Data type	Description
a_rc	sqlint32*	Return code; 0 indicates success.

Return value

None.

get_connection function

Where defined

Remote_Operation

Purpose

Retrieve a pointer to the connection that owns this operation.

Syntax

```
Remote_Connection* get_connection ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the Remote_Connection object.

get_server function

Where defined

Remote_Operation

Purpose

Retrieve a pointer to the server that owns this operation.

Syntax

Fenced_Generic_Server* get_server ()

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server object.

get_input_data function**Where defined**

Remote_Operation

Purpose

Retrieve a pointer to the list of input values to the operation.

Syntax

Runtime_Data_List* get_input_data ()

Input arguments

None.

Output arguments

None.

Return value

Pointer to the list of data values.

get_output_data function**Where defined**

Remote_Operation

Purpose

Retrieve a pointer to the list of output data buffers.

Syntax

Runtime_Data_List* get_output_data ()

Input arguments

None.

Output arguments

None.

Return value

Pointer to the list of data buffers.

Remote_Query

get_exec_desc function

Where defined

Remote_Operation

Purpose

Retrieve a pointer and the length for the execution descriptor for this query.

Syntax

```
void get_exec_desc (void** a_exec_desc,  
                  int*   a_exec_desc_len)
```

Input arguments

None.

Output arguments

Table 221. Output arguments for the get_exec_desc member function

Name	Data type	Description
a_exec_desc	void**	Pointer to the execution descriptor.
a_exec_desc_len	int*	Length of the descriptor.

Return value

None.

open function

Purpose

Open a query.

Usage DB2 Information Integrator invokes this member function to initiate a query. This member function must be implemented by the wrapper in the wrapper-specific Remote_Query subclass.

Syntax

```
sqlint32 open ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

reopen function**Purpose**

Reopen a query

Usage DB2 Information Integrator invokes this member function to initiate a query with new parameter values. This member function must be implemented by the wrapper in the wrapper-specific Remote_Query subclass.

Syntax

```
sqlint32 reopen (sqlint16 a_status)
```

Input arguments

Table 222. Input arguments for the reopen member function

Name	Data type	Description
a_status	sqlint16	Unused.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

fetch function**Purpose**

Fetch a single row.

Usage DB2 Information Integrator invokes this member function to fetch a row of non-LOB data from a remote query. This member function must be implemented by the wrapper in the wrapper-specific Remote_Query subclass.

Syntax

```
sqlint32 fetch ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Remote_Query

close function

Purpose

Close a query.

Usage DB2 Information Integrator invokes this member function to close a query cursor. This member function must be implemented by the wrapper in the wrapper-specific Remote_Query subclass.

Syntax

```
sqlint32 close (sqlint16 a_status)
```

Input arguments

Table 223. Input arguments for the close member function

Name	Data type	Description
a_status	sqlint16	Status (SQLQG_CLOSE_EOS, SQLQG_CLOSE_EOA, or SQLQG_CLOSE_EOQ). SQLQG_CLOSE_EOS indicates that the wrapper leaves the state so that the reopen() function can be called. SQLQG_CLOSE_EOA indicates that the wrapper can finish all necessary processing. SQLQG_CLOSE_EOQ is not used.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

report_eof function

Purpose

Report an end-of-file condition during a fetch.

Usage This member function is invoked by the wrapper-specific Remote_Query class during the fetch() method to indicate that the last row was fetched.

Syntax

```
sqlint32 report_eof ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. This code is returned from fetch().

get_status function**Purpose**

Retrieve the current query status.

Syntax

sqluint8 get_status ()

Input arguments

None.

Output arguments

None.

Return value

Status (SQLQG_UNREADY, SQLQG_READY, SQLQG_OPEN, or SQLQG_EOF).

set_status function**Purpose**

Set the current query status.

Syntax

void set_status (sqluint8 a_new_status)

Input arguments*Table 224. Input arguments for the set_status member function*

Name	Data type	Description
a_new_status	sqluint8	Status (SQLQG_UNREADY, SQLQG_READY, SQLQG_OPEN, or SQLQG_EOF).

Output arguments

None.

Remote_Query

Return value
None.

fetch_lob function

Purpose

Retrieve LOB data into a special LOB buffer that is provided by DB2.

Syntax

```
virtual sqlint32 fetch_lob (unsigned char* a_buffer,  
                           sqluint32      a_buffer_size,  
                           sqluint32      a_bytes_written)
```

Input arguments

Table 225. Input arguments for the fetch_lob member function

Name	Data type	Description
a_buffer	unsigned char*	Address of the LOB data buffer that is provided by DB2.
a_buffer_size	sqluint32	Size (in bytes) of the buffer.
a_bytes_written	sqluint32	Total number of bytes that were written for the current column by previous calls to the fetch_lob function.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

set_lob_next function

Purpose

Suspend the processing of non-LOB columns and start the processing of one or more LOB columns. This function is called from the fetch function.

Syntax

```
void set_lob_next (void)
```

Input arguments

None.

Output arguments

None.

Return value

None.

lob_data_ready function**Purpose**

Indicates that the wrapper transferred a section of LOB data to the buffer.

Syntax

```
sqlint32 lob_data_ready (sqlint32      a_col_number,
                        sqluint32     a_bytes_ready,
                        sqlqg_lob_status a_status,
                        sqlqg_lob_intent a_next)
```

Input arguments

Table 226. Input arguments for the lob_data_ready member function

Name	Data type	Description
a_col_number	sqlint32	Column from which the wrapper took data to write to the LOB buffer.
a_bytes_ready	sqluint32	Number of bytes that the wrapper copied to the LOB buffer.
a_status	sqlqg_lob_status	Enumerated data type that indicates the status. The value is LOB_LAST if the wrapper is processing the last section of the column. Otherwise, the value is LOB_MORE.
a_next	sqlqg_lob_intent	Enumerated data type that indicates whether the wrapper's next call is to the fetch or to the fetch_lob function. Valid values are LOB_NEXT, NON_LOB_NEXT, and ROW_DONE.

Output arguments

None.

Remote_Query

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Operation classes for the C++ API” on page 131

Remote_Passthru class

This topic describes the Remote_Passthru class and provides details for the constructor and the member functions.

Overview

The Remote_Passthru class represents a pass-through operation at a remote data source. A wrapper must subclass the Remote_Passthru class to support pass-through operations.

The Remote_Passthru class is one of the operation classes for the C++ API.

Usage This class is instantiated by the create_remote_passthru() method of the wrapper-specific Remote_Connection subclass.

File sqlqg_operation.h

Parent class

Remote_Operation

Data members

None.

Constructors and member functions

The following tables describe the constructor and the member functions of the Remote_Passthru class. The constructor and functions are described in more detail after the tables.

Table 227. Constructors for the Remote_Passthru class

Constructor	Description
Remote_Passthru	Construct a Remote_Passthru object.

Table 228. Member functions for the Remote_Passthru class

Member function	Description
get_connection	Retrieve a pointer to the connection that owns this operation.
get_server	Retrieve a pointer to the server that owns this operation.

Table 228. Member functions for the Remote_Passthru class (continued)

Member function	Description
get_input_data	Retrieve a pointer to the list of input values to the operation.
get_output_data	Retrieve a pointer to the list of output data buffers.
get_SQL_statement	Retrieve the statement for pass-through execution.
report_eof	Report an end-of-file condition during a fetch.
prepare	Prepare a remote pass-through operation.
describe	Describe a remote pass-through operation.
execute	Run a remote pass-through operation.
open	Open a cursor for a remote pass-through operation.
fetch	Fetch a row in a remote pass-through operation.
close	Close a cursor for a remote pass-through operation.

Remote_Passthru constructor

Purpose

Construct a Remote_Passthru object.

Syntax

```
Remote_Passthru (Remote_Connection* a_active_connection,
                Runtime_Operation* a_runtime_passthru,
                sqlint32*          a_rc)
```

Input arguments

Table 229. Input arguments for the Remote_Passthru constructor

Name	Data type	Description
a_active_connection	Remote_Connection*	Connection for this query.
a_runtime_info	Runtime_Operation*	Internal data for the query.

Remote_Passthru

Output arguments

Table 230. Output arguments for the Remote_Passthru constructor

Name	Data type	Description
a_rc	sqlint32*	Return code; 0 indicates success.

Return value

None.

get_connection function

Where defined

Remote_Operation

Purpose

Retrieve a pointer to the connection that owns this operation.

Syntax

```
Remote_Connection* get_connection ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the Remote_Connection object.

get_server function

Where defined

Remote_Operation

Purpose

Retrieve a pointer to the server that owns this operation.

Syntax

```
Fenced_Generic_Server* get_server ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server object.

get_input_data function**Where defined**

Remote_Operation

Purpose

Retrieve a pointer to the list of input values to the operation.

Syntax

```
Runtime_Data_List* get_input_data ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the list of data values.

get_output_data function**Where defined**

Remote_Operation

Purpose

Retrieve a pointer to the list of output data buffers.

Syntax

```
Runtime_Data_List* get_output_data ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the list of data buffers.

get_SQL_statement function**Where defined**

Remote_Operation

Remote_Passthru

Purpose

Retrieve the statement for pass-through execution.

Syntax

```
char* get_SQL_statement ()
```

Input arguments

None.

Output arguments

None.

Return value

Statement (null-terminated).

report_eof function

Where defined

Remote_Operation

Purpose

Report an end-of-file condition during a fetch.

Syntax

```
sqlint32 report_eof ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. This code is returned from fetch().

prepare function

Purpose

Prepare a remote pass-through operation.

Syntax

```
sqlint32 prepare (Runtime_Data_Desc_List* a_data_description_list)
```

Input arguments

None.

Output arguments*Table 231. Output arguments for the prepare member function*

Name	Data type	Description
a_data_description_list	Runtime_Data_Desc_List*	A list of Runtime_Data_Desc that describes the results of the pass-through operation.

Return value

Return code. A value of 0 indicates success.

describe function**Purpose**

Describe a remote pass-through operation.

Syntax

```
sqlint32 describe (Runtime_Data_Desc_List* a_data_description_list)
```

Input arguments

None.

Output arguments*Table 232. Output arguments for the describe member function*

Name	Data type	Description
a_data_description_list	Runtime_Data_Desc_List*	A list of Runtime_Data_Desc that describes the results of the pass-through operation.

Return value

Return code. A value of 0 indicates success.

execute function**Purpose**

Run a remote pass-through operation.

Syntax

```
sqlint32 execute ()
```

Input arguments

None.

Remote_Passthru

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

open function

Purpose

Open a cursor for a remote pass-through operation.

Syntax

```
sqlint32 open ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

fetch function

Purpose

Fetch a row in a remote pass-through operation.

Syntax

```
sqlint32 fetch ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

close function

Purpose

Close a cursor for a remote pass-through operation.

Syntax

```
sqlint32 close ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Operation classes for the C++ API” on page 131

Request classes for the C++ API

The following table describes each request class for the C++ API.

Table 233. Request classes

Class name	Description
Request	This class encapsulates a query fragment to be analyzed and processed by the wrapper.
Reply	This class represents a part of a query that can be processed by the wrapper. This class can be subclassed if the wrapper uses a cost model other than the default.
Request_Exp	This class describes an SQL expression node. The SQL expression node can be a head expression (select list) or part of a predicate.
Request_Exp_Type	This class represents a data type descriptor for Request_Exp nodes.
Request_Constant	This class represents a data value for a Request_Exp node with a type of Request_Exp::constant.
Predicate_List	This class encapsulates a list of predicates and is used by the RRC protocol.

Related reference:

- “Request_Exp class” on page 173
- “Request class” on page 150
- “Request_Constant class” on page 184
- “Request_Exp_Type class” on page 180
- “Reply class” on page 156
- “Predicate_List class” on page 188

Request class

This topic describes the Request class and provides details for each member function.

Overview

The Request class encapsulates a query fragment to be analyzed and processed by the wrapper.

The Request class is one of the request classes for the C++ API.

Usage This class is never instantiated by the wrapper.

File sqlqg_request.h

Parent class

Parsed_Query_Fragment

Data members

None.

Member functions

The following table describes each member function of the Request class. Each function is described in more detail after the table.

Table 234. Member functions for the Request class

Member function	Description
get_number_of_quantifiers	Retrieve the number of quantifiers for this query.
get_number_of_predicates	Retrieve the number of predicates for this query.
get_number_of_head_exp	Retrieve the number of head (select list) expressions for this query.
get_quantifier_handle	Retrieve the handle for the <i>n</i> th quantifier for this query.
get_predicate_handle	Retrieve the handle for the <i>n</i> th predicate for this query.
get_head_exp_handle	Retrieve the handle for the <i>n</i> th head expression for this query.
get_nickname	Retrieve the nickname object that is associated with the specified handle.
get_head_exp	Retrieve a head expression that is associated with the specified handle.
get_predicate	Retrieve a predicate expression that is associated with the specified handle.

Table 234. Member functions for the Request class (continued)

Member function	Description
get_distinct	Retrieve the DISTINCT flag for a query.

get_number_of_quantifiers function**Where defined**

Parsed_Query_Fragment

Purpose

Retrieve the number of quantifiers (nicknames and table expressions) for this query.

Syntax

```
int get_number_of_quantifiers ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of quantifiers.

get_number_of_predicates function**Where defined**

Parsed_Query_Fragment

Purpose

Retrieve the number of predicates for this query.

Syntax

```
int get_number_of_predicates ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of predicates.

get_number_of_head_exp function

Where defined

Parsed_Query_Fragment

Purpose

Retrieve the number of head (select list) expressions for this query.

Syntax

```
int get_number_of_head_exp ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of head expressions.

get_quantifier_handle function

Where defined

Parsed_Query_Fragment

Purpose

Retrieve the handle for the n th quantifier for this query.

Syntax

```
sqlint32 get_quantifier_handle (int a_quant_pos,  
                               int* a_quant_handle)
```

Input arguments

Table 235. Input arguments for the get_quantifier_handle member function

Name	Data type	Description
a_quant_pos	int	Position of the handle in list (starting at 1).

Output arguments

Table 236. Output arguments for the get_quantifier_handle member function

Name	Data type	Description
a_quant_handle	int*	Handle.

Return value

Return code. A value of 0 indicates success.

get_predicate_handle function**Where defined**

Parsed_Query_Fragment

Purpose

Retrieve the handle for the *n*th predicate for this query.

Syntax

```
sqlint32 get_predicate_handle (int a_pred_pos,
                              int* a_pred_handle )
```

Input arguments

Table 237. Input arguments for the get_predicate_handle member function

Name	Data type	Description
a_pred_pos	int	Position of the handle in list (starting at 1).

Output arguments

Table 238. Output arguments for the get_predicate_handle member function

Name	Data type	Description
a_pred_handle	int*	Handle.

Return value

Return code. A value of 0 indicates success.

get_head_exp_handle function**Where defined**

Parsed_Query_Fragment

Purpose

Retrieve the handle for the *n*th head expression for this query.

Syntax

```
sqlint32 get_head_exp_handle (int a_head_exp_pos,
                              int* a_head_exp_handle)
```

Input arguments

Table 239. Input arguments for the get_head_exp_handle member function

Name	Data type	Description
a_head_exp_pos	int	Position of the handle in list (starting at 1).

Output arguments

Table 240. Output arguments for the `get_head_exp_handle` member function

Name	Data type	Description
<code>a_head_exp_handle</code>	<code>int*</code>	Handle.

Return value

Return code. A value of 0 indicates success.

`get_nickname` function

Where defined

`Parsed_Query_Fragment`

Purpose

Retrieve the nickname object that is associated with the specified handle.

Syntax

```
sqlint32 get_nickname (int a_quant_handle,  
Unfenced_Generic_Nickname** a_nickname)
```

Input arguments

Table 241. Input arguments for the `get_nickname` member function

Name	Data type	Description
<code>a_quant_handle</code>	<code>int</code>	Handle for the nickname.

Output arguments

Table 242. Output arguments for the `get_nickname` member function

Name	Data type	Description
<code>a_nickname</code>	<code>Unfenced_Generic_Nickname**</code>	Pointer to the nickname object.

Return value

Return code. A value of 0 indicates success.

`get_head_exp` function

Where defined

`Parsed_Query_Fragment`

Purpose

Retrieve a head expression that is associated with the specified handle.

Syntax

```
sqlint32 get_head_exp (int          a_head_exp_handle,
                      Request_Exp** a_head_exp)
```

Input arguments

Table 243. Input arguments for the get_head_exp member function

Name	Data type	Description
a_head_exp_handle	int	Handle for the expression.

Output arguments

Table 244. Output arguments for the get_head_exp member function

Name	Data type	Description
a_head_exp	Request_Exp**	Pointer to the expression object.

Return value

Return code. A value of 0 indicates success.

get_predicate function**Where defined**

Parsed_Query_Fragment

Purpose

Retrieve a predicate expression that is associated with the specified handle.

Syntax

```
sqlint32 get_predicate (int          a_pred_handle,
                      Request_Exp** a_pred_exp)
```

Input arguments

Table 245. Input arguments for the get_predicate member function

Name	Data type	Description
a_pred_handle	int	Handle for the expression.

Request

Output arguments

Table 246. Output arguments for the `get_predicate` member function

Name	Data type	Description
<code>a_pred_exp</code>	<code>Request_Exp**</code>	Pointer to the expression object.

Return value

Return code. A value of 0 indicates success.

`get_distinct` function

Where defined

`Parsed_Query_Fragment`

Purpose

Retrieve the DISTINCT flag for a query.

Syntax

```
int get_distinct ()
```

Input arguments

None.

Output arguments

None.

Return value

Distinct flag.

Related reference:

- “Request classes for the C++ API” on page 149

Reply class

This topic describes the Reply class and provides details for the constructor and the member functions.

Overview

The Reply class represents a portion of a query that the wrapper can process. This class can be subclassed if the wrapper uses a cost model other than the default model.

The Reply class is one of the request classes for the C++ API.

Usage This class is instantiated by the `create_reply()` method of the `Unfenced_Generic_Server` class. If the wrapper implements a subclass

of the Reply class, the create_reply() method is overridden in the wrapper-specific subclass of Unfenced_Generic_Server.

File sqlqg_request.h

Parent class
Parsed_Query_Fragment

Data members

None.

Constructors and member functions

The following tables describe the constructor and the member functions of the Reply class. The constructor and functions are described in more detail after the tables.

Table 247. Constructors for the Reply class

Constructor	Description
Reply	Construct an empty reply object.

Table 248. Member functions for the Reply class

Member function	Description
get_number_of_quantifiers	Retrieve the number of quantifiers for this query.
get_number_of_predicates	Retrieve the number of predicates for this query.
get_number_of_head_exp	Retrieve the number of head (select list) expressions for this query.
get_quantifier_handle	Retrieve the handle for the <i>n</i> th quantifier for this query.
get_predicate_handle	Retrieve the handle for the <i>n</i> th predicate for this query.
get_head_exp_handle	Retrieve the handle for the <i>n</i> th head expression for this query.
get_nickname	Retrieve the nickname object that is associated with the specified handle.
get_head_exp	Retrieve a head expression that is associated with the specified handle.
get_predicate	Retrieve a predicate expression that is associated with the specified handle.
get_distinct	Retrieve the DISTINCT flag for a query.
set_distinct	Set the DISTINCT flag for a query.
add_head_exp	Add a head expression to the reply.

Reply

Table 248. Member functions for the Reply class (continued)

Member function	Description
add_predicate	Add a predicate expression to the reply.
add_quantifier	Add a quantifier to the reply.
add_order_entry	Add an ORDER BY specification to the reply.
get_number_of_order_entries	Retrieve the number of ORDER BY entries for the reply.
get_order_entry	Retrieve a specific ORDER BY entry.
get_exec_desc	Retrieve the execution descriptor that is associated with the reply.
set_exec_desc	Store an execution descriptor in the reply.
get_parameter_order	Retrieve a list of parameter handles.
cardinality	Retrieve the cardinality for the query fragment.
total_cost	Retrieve the total cost for a query fragment.
re_exec_cost	Cost to run a query fragment again.
first_tuple_cost	Retrieve the cost to fetch the first tuple.
all_costs	Retrieve all costing values at one time.
next	Retrieve a pointer to the next reply in a chain of replies.
set_next_reply	Link a new reply to the current reply.
get_server	Retrieve a pointer to the server that owns the reply.

Reply constructor

Purpose

Construct an empty reply object.

Syntax

```
Reply (Request*          a_rq,  
       Unfenced_Generic_Server* a_server,  
       sqlint32*         a_rc)
```

Input arguments*Table 249. Input arguments for the Reply constructor*

Name	Data type	Description
a_rq	Request*	Request from which this reply is derived.
a_server	Unfenced_Generic_Server*	Server for a request or reply protocol.

Output arguments*Table 250. Output arguments for the Reply constructor*

Name	Data type	Description
a_rc	sqlint32*	Return code; 0 indicates success.

Return value

None.

get_number_of_quantifiers function**Where defined**

Parsed_Query_Fragment

Purpose

Retrieve the number of quantifiers (nicknames and table expressions) for this query.

Syntax

int get_number_of_quantifiers ()

Input arguments

None.

Output arguments

None.

Return value

Number of quantifiers.

get_number_of_predicates function**Where defined**

Parsed_Query_Fragment

Reply

Purpose

Retrieve the number of predicates for this query.

Syntax

```
int get_number_of_predicates ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of predicates.

get_number_of_head_exp function

Where defined

Parsed_Query_Fragment

Purpose

Retrieve the number of head (select list) expressions for this query.

Syntax

```
int get_number_of_head_exp ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of head expressions.

get_quantifier_handle function

Where defined

Parsed_Query_Fragment

Purpose

Retrieve the handle for the *n*th quantifier for this query.

Syntax

```
sqlint32 get_quantifier_handle (int a_quant_pos,  
                               int* a_quant_handle)
```


Input arguments

Table 251. Input arguments for the `get_quantifier_handle` member function

Name	Data type	Description
<code>a_quant_pos</code>	<code>int</code>	Position of the handle in list (starting at 1).

Output arguments

Table 252. Output arguments for the `get_quantifier_handle` member function

Name	Data type	Description
<code>a_quant_handle</code>	<code>int*</code>	Handle.

Return value

Return code. A value of 0 indicates success.

`get_predicate_handle` function

Where defined

`Parsed_Query_Fragment`

Purpose

Retrieve the handle for the *n*th predicate for this query.

Syntax

```
sqlint32 get_predicate_handle (int a_pred_pos,
                              int* a_pred_handle)
```

Input arguments

Table 253. Input arguments for the `get_predicate_handle` member function

Name	Data type	Description
<code>a_pred_pos</code>	<code>int</code>	Position of the handle in list (starting at 1).

Output arguments

Table 254. Output arguments for the `get_predicate_handle` member function

Name	Data type	Description
<code>a_pred_handle</code>	<code>int*</code>	Handle.

Return value

Return code. A value of 0 indicates success.

get_head_exp_handle function

Where defined

Parsed_Query_Fragment

Purpose

Retrieve the handle for the *n*th head expression for this query.

Syntax

```
sqlint32 get_head_exp_handle (int a_head_exp_pos,
                             int* a_head_exp_handle)
```

Input arguments

Table 255. Input arguments for the *get_head_exp_handle* member function

Name	Data type	Description
a_head_exp_pos	int	Position of the handle in list (starting at 1).

Output arguments

Table 256. Output arguments for the *get_head_exp_handle* member function

Name	Data type	Description
a_head_exp_handle	int*	Handle.

Return value

Return code. A value of 0 indicates success.

get_nickname function

Where defined

Parsed_Query_Fragment

Purpose

Retrieve the nickname object that is associated with the specified handle.

Syntax

```
sqlint32 get_nickname (int a_quant_handle,
                      Unfenced_Generic_Nickname** a_nickname)
```

Input arguments

Table 257. Input arguments for the *get_nickname* member function

Name	Data type	Description
a_quant_handle	int	Handle for the nickname.

Output arguments*Table 258. Output arguments for the get_nickname member function*

Name	Data type	Description
a_nickname	Unfenced_Generic_Nickname**	Pointer to the nickname object.

Return value

Return code. A value of 0 indicates success.

get_head_exp function**Where defined**

Parsed_Query_Fragment

Purpose

Retrieve a head expression that is associated with the specified handle.

Syntax

```
sqlint32 get_head_exp (int a          a_head_exp_handle,
                      Request_Exp** a_head_exp)
```

Input arguments*Table 259. Input arguments for the get_head_exp member function*

Name	Data type	Description
a_head_exp_handle	int	Handle for the expression.

Output arguments*Table 260. Output arguments for the get_head_exp member function*

Name	Data type	Description
a_head_exp	Request_Exp**	Pointer to the expression object.

Return value

Return code. A value of 0 indicates success.

get_predicate function**Where defined**

Parsed_Query_Fragment

Reply

Purpose

Retrieve a predicate expression that is associated with the specified handle.

Syntax

```
sqlint32 get_predicate (int          a_pred_handle,  
                       Request_Exp** a_pred_exp)
```

Input arguments

Table 261. Input arguments for the `get_predicate` member function

Name	Data type	Description
<code>a_pred_handle</code>	<code>int</code>	Handle for the expression.

Output arguments

Table 262. Output arguments for the `get_predicate` member function

Name	Data type	Description
<code>a_pred_exp</code>	<code>Request_Exp**</code>	Pointer to the expression object.

Return value

Return code. A value of 0 indicates success.

`get_distinct` function

Where defined

`Parsed_Query_Fragment`

Purpose

Retrieve the `DISTINCT` flag for a query.

Syntax

```
int get_distinct ()
```

Input arguments

None.

Output arguments

None.

Return value

Distinct flag.

set_distinct function**Where defined**

Parsed_Query_Fragment

Purpose

Set the DISTINCT flag for a query.

Syntax

```
void set_distinct (int a_distinct)
```

Input arguments

Table 263. Input arguments for the set_distinct member function

Name	Data type	Description
a_distinct	int	Distinct flag (1 for DISTINCT; 0 for not DISTINCT).

Output arguments

None.

Return value

None.

add_head_exp function**Purpose**

Add a head expression to the reply.

Syntax

```
sqlint32 add_head_exp (int a_head_exp_handle)
```

Input arguments

Table 264. Input arguments for the add_head_exp member function

Name	Data type	Description
a_head_exp_handle	int	Handle for the expression.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Reply

add_predicate function

Purpose

Add a predicate expression to the reply.

Syntax

```
sqlint32 add_predicate_exp (int a_pred_handle)
```

Input arguments

Table 265. Input arguments for the add_predicate member function

Name	Data type	Description
a_pred_handle	int	Handle for the expression.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

add_quantifier function

Purpose

Add a quantifier to the reply.

Syntax

```
sqlint32 add_quantifier (int a_quant_handle)
```

Input arguments

Table 266. Input arguments for the add_quantifier member function

Name	Data type	Description
a_quant_handle	int	Handle for the quantifier.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

add_order_entry function

Purpose

Add an ORDER BY specification to the reply.

Syntax

```
sqlint32 add_order_entry (int a_gindex,
                          Reply::order_direction a_direction)
```

Input arguments

Table 267. Input arguments for the `add_order_entry` member function

Name	Data type	Description
<code>a_gindex</code>	<code>int</code>	Head expression index for ordering (not a handle).
<code>a_direction</code>	<code>Reply_order_direction</code>	Direction (ascending or descending) for ordering.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

`get_number_of_order_entries` function

Purpose

Retrieve the number of ORDER BY entries for the reply.

Syntax

```
int get_number_of_order_entries ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of ORDER BY entries.

`get_order_entry` function

Purpose

Retrieve a specific ORDER BY entry.

Syntax

```
get_order_entry (int a_pos,
                 int* a_gindexP,
                 Reply::order_direction* a_direction)
```

Reply

Input arguments

Table 268. Input arguments for the `get_order_entry` member function

Name	Data type	Description
<code>a_pos</code>	<code>int</code>	Position of order entry (starting at 1).

Output arguments

Table 269. Output arguments for the `get_order_entry` member function

Name	Data type	Description
<code>a_gindexP</code>	<code>int*</code>	Head expression index for the ORDER BY expression.
<code>a_direction</code>	<code>Reply::order_direction*</code>	Direction for ordering.

Return value

Return code. A value of 0 indicates success.

`get_exec_desc` function

Purpose

Retrieve the execution descriptor that is associated with the reply.

Syntax

```
void get_exec_desc (void** a_exec_desc,  
                   int*   a_exec_desc_size)
```

Input arguments

None.

Output arguments

Table 270. Output arguments for the `get_exec_desc` member function

Name	Data type	Description
<code>a_exec_desc</code>	<code>void**</code>	Pointer to the execution descriptor.
<code>a_exec_desc_size</code>	<code>int*</code>	Size of the execution descriptor.

Return value

None.

set_exec_desc function**Purpose**

Store an execution descriptor in the reply.

Usage Storage for the execution descriptor must be allocated with `Wrapper_Utillities::allocate`.

Syntax

```
void set_exec_desc (void** a_exec_desc,
                  int*   a_exec_desc_size)
```

Input arguments

Table 271. Input arguments for the set_exec_desc member function

Name	Data type	Description
a_exec_desc	void**	Pointer to the execution descriptor.
a_exec_desc_size	int*	Size of the execution descriptor.

Output arguments

None.

Return value

None.

get_parameter_order function**Purpose**

Retrieve a list of parameter handles. The order of the list corresponds to the order of the parameter values in the `Remote_Operation` object.

Syntax

```
sqlint32 get_parameter_order (int*   a_number_of_params,
                              int**  a_param_handle_array)
```

Input arguments

None.

Output arguments

Table 272. Output arguments for the get_parameter_order member function

Name	Data type	Description
a_number_of_params	int*	Number of the parameter handles.
a_param_handle_array	int**	Array of the parameter handles.

Return value

Return code. A value of 0 indicates success.

cardinality function**Purpose**

Retrieve the cardinality for the query fragment.

Syntax

```
sqlint32 cardinality (float* a_cardinality)
```

Input arguments

None.

Output arguments

Table 273. Output arguments for the cardinality member function

Name	Data type	Description
a_cardinality	float*	Cardinality.

Return value

Return code. A value of 0 indicates success.

total_cost function**Purpose**

Retrieve the total cost for a query fragment.

Syntax

```
sqlint32 total_cost (float* a_total_cost)
```

Input arguments

None.

Output arguments

Table 274. Output arguments for the total_cost member function

Name	Data type	Description
a_total_cost	float*	Cost.

Return value

Return code. A value of 0 indicates success.

re_exec_cost function**Purpose**

Cost to run a query fragment again.

Syntax

```
sqlint32 re_exec_cost (float* a_re_exec_cost)
```

Input arguments

None.

Output arguments

Table 275. Output arguments for the re_exec_cost member function

Name	Data type	Description
a_re_exec_cost	float*	Cost.

Return value

Return code. A value of 0 indicates success.

first_tuple_cost function**Purpose**

Retrieve the cost to fetch the first tuple.

Syntax

```
sqlint32 first_tuple_cost (float* a_first_tuple_cost)
```

Input arguments

None.

Output arguments

Table 276. Output arguments for the first_tuple_cost member function

Name	Data type	Description
a_first_tuple_cost	float*	Cost.

Return value

Return code. A value of 0 indicates success.

all_costs function**Purpose**

Retrieve all costing values at one time.

Syntax

Reply

```
sqlint32 all_costs (float* a_total_cost,  
                  float* a_first_tuple_cost,  
                  float* a_re_exec_cost)
```

Input arguments

None.

Output arguments

Table 277. Output arguments for the all_costs member function

Name	Data type	Description
a_total_cost	float*	Total cost.
a_first_tuple_cost	float*	First tuple cost.
a_re_exec_cost	float*	Re-execute cost.

Return value

Return code. A value of 0 indicates success.

next function

Purpose

Retrieve a pointer to the next reply in a chain of replies.

Syntax

```
Reply* next ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the next reply. The value is null if at the end of the chain.

set_next_reply function

Purpose

Link a new reply to the current reply.

Usage The wrapper verifies that it adds the reply to the end of the chain. Otherwise, memory leaks can occur.

Syntax

```
void set_next_reply (Reply* a_next_reply)
```

Input arguments*Table 278. Input arguments for the set_next_reply member function*

Name	Data type	Description
a_next_reply	Reply*	Reply to be added to a chain.

Output arguments

None.

Return value

None.

get_server function**Purpose**

Retrieve a pointer to the server that owns this reply.

Syntax

Unfenced_Generic_server* get_server ()

Input arguments

None.

Output arguments

None.

Return value

Pointer to the server object.

Related reference:

- “Request classes for the C++ API” on page 149

Request_Exp class

This topic describes the Request_Exp class and provides details for each member function.

Overview

The Request_Exp class represents a node in an expression tree. This node can be a column reference, a constant value, a host parameter, or an operator.

The Request_Exp class is one of the request classes for the C++ API.

Usage This class is never instantiated by the wrapper.

File sqlqg_request.h

Request_Exp

Data members

None.

Types Request_Exp::kind

Type enum

Values

badkind, column, unbound, constant, oper

Member functions

The following table describes each member function of the Request_Exp class. Each function is described in more detail after the table.

Table 279. Member functions for the Request_Exp class

Member function	Description
get_kind	Retrieve the kind of expression node.
get_data_type	Retrieve the data type that is associated with the expression.
get_parent	Retrieve a pointer to the parent expression node for the current node.
get_next_child	Retrieve a pointer to the next child of the same parent.
get_handle	Retrieve the handle for the expression node.
get_column_name	Retrieve the column name for a column expression node.
get_column_quantifier_handle	Retrieve the handle to the column quantifier.
get_value	Retrieve the value of a constant expression node.
get_first_child	Retrieve the first child of an operator expression node.
get_number_of_children	Retrieve the number of children for an operator expression node.
get_token	Retrieve the token for an operator expression node.
get_signature	Retrieve the full signature for an operator expression node.

get_kind function

Purpose

Retrieve the kind of expression node.

Usage An additional syntax for this member function is `sqlint32 get_kind()`. This syntax is used in the `Unfenced_Server::get_selectivity()` expression when a Request or Reply is not available. This `get_kind()` syntax does not distinguish between column references and unbound references.

Syntax

```
sqlint32 get_kind (Parsed_Query_Fragment* a_query_fragment,
                  kind* a_kind )
```

Input arguments

Table 280. Input arguments for the `get_kind` member function

Name	Data type	Description
<code>a_query_fragment</code>	<code>Parsed_Query_Fragment*</code>	Request or Reply to which this expression belongs.

Output arguments

Table 281. Output arguments for the `get_kind` member function

Name	Data type	Description
<code>a_kind</code>	<code>kind*</code>	Kind of expression node.

Return value

Return code. A value of 0 indicates success.

get_data_type function

Purpose

Retrieve the data type that is associated with the expression.

Syntax

```
sqlint32 get_data_type (Request_Exp_Type** a_new_type)
```

Input arguments

None.

Output arguments

Table 282. Output arguments for the `get_data_type` member function

Name	Data type	Description
<code>a_new_type</code>	<code>Request_Exp_Type**</code>	Pointer to the type of descriptor.

Return value

Return code. A value of 0 indicates success.

get_parent function

Purpose

Retrieve a pointer to the parent expression node for the current node.

Syntax

```
Request_Exp* get_parent ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the parent node. The value is null if this is a top-level node.

get_next_child function

Purpose

Retrieve a pointer to the next child of the same parent (that is, the next sibling of the current node).

Syntax

```
Request_Exp* get_next_child ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the next child. The value is null if there are no more siblings.

get_handle function

Purpose

Retrieve the handle for the expression node.

Syntax

```
sqlint32 get_handle (int* a_handle)
```

Input arguments

None.

Output arguments*Table 283. Output arguments for the get_handle member function*

Name	Data type	Description
a_handle	int*	Handle.

Return value

Return code. A value of 0 indicates success.

get_column_name function**Purpose**

Retrieve the column name for a column expression node.

Usage This method is valid only when the expression kind is Request_Exp::column.

Syntax

```
sqlint32 get_column_name (char** a_col_name,
                          int*   a_name_length)
```

Input arguments

None.

Output arguments*Table 284. Output arguments for the get_column_name member function*

Name	Data type	Description
a_col_name	char**	Pointer to the column name (not null-terminated).
a_name_length	int*	Length of the column name.

Return value

Return code. A value of 0 indicates success.

get_column_quantifier_handle function**Purpose**

Retrieve the handle to the column quantifier (the table or the table expression).

Usage This method is valid only when the expression kind is Request_Exp::column.

Syntax

Request_Exp

```
sqlint32 get_column_quantifier_handle (int* a_quant_handle)
```

Input arguments

None.

Output arguments

Table 285. Output arguments for the get_column_quantifier_handle member function

Name	Data type	Description
a_quant_handle	int*	Quantifier handle.

Return value

Return code. A value of 0 indicates success.

get_value function

Purpose

Retrieve the value of a constant expression node.

Usage This method is valid only when the expression kind is Request_Exp::constant.

Syntax

```
sqlint32 get_value (Request_Constant** a_constant_value)
```

Input arguments

None.

Output arguments

Table 286. Output arguments for the get_value member function

Name	Data type	Description
a_constant_value	Request_Constant**	Pointer to the constant value.

Return value

Return code. A value of 0 indicates success.

get_first_child function

Purpose

Retrieve the first child of an operator expression node.

Usage This method is valid only when the expression kind is Request_Exp::oper.

Syntax

```
Request_Exp* get_first_child ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the first child. The value is null if there are no children.

get_number_of_children function**Purpose**

Retrieve the number of children for an operator expression node.

Usage This method is valid only when the expression kind is Request_Exp::oper.

Syntax

```
int get_number_of_children ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of children.

get_token function**Purpose**

Retrieve the token (the function name) for an operator expression node.

Usage This method is valid only when the expression kind is Request_Exp::oper.

Syntax

```
sqlint32 get_token (char** a_operator_token,
                   int* a_token_length )
```

Input arguments

None.

Request_Exp

Output arguments

Table 287. Output arguments for the `get_token` member function

Name	Data type	Description
<code>a_operator_token</code>	<code>char**</code>	Pointer to the token (not null-terminated).
<code>a_token_length</code>	<code>int*</code>	Length of the token.

Return value

Return code. A value of 0 indicates success.

`get_signature` function

Purpose

Retrieve the full signature for an operator expression node.

Usage This method is valid only when the expression kind is `Request_Exp::oper`.

Syntax

```
sqlint32 get_signature (char** a_signature)
```

Input arguments

None.

Output arguments

Table 288. Output arguments for the `get_signature` member function

Name	Data type	Description
<code>a_signature</code>	<code>char**</code>	Pointer to the null-terminated signature.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Request classes for the C++ API” on page 149

Request_Exp_Type class

This topic describes the `Request_Exp_Type` class and provides details for each member function.

Overview

The Request_Exp_Type class represents a data type descriptor for Request_Exp nodes.

The Request_Exp_Type class is one of the request classes for the C++ API.

Usage This class is never instantiated by the wrapper.

File sqlqg_runtime_data_operation.h

Data members

None.

Member functions

The following table describes each member function of the Request_Exp_Type class. Each function is described in more detail after the table.

Table 289. Member functions for the Request_Exp_Type class

Member function	Description
get_for_bit_data	Retrieve the FOR BIT DATA flag.
get_null_indicator	Retrieve the nullable indicator.
get_data_type	Retrieve the data type.
get_maximum_length	Retrieve the maximum length for the type.
get_precision	Get the precision for numeric types.
get_scale	Get the scale for a numeric type.
get_codepage	Retrieve the code page for character types.

get_for_bit_data function

Purpose

Retrieve the FOR BIT DATA flag for the data.

Syntax

```
unsigned char get_for_bit_data ()
```

Input arguments

None.

Output arguments

None.

Return value

FOR BIT DATA flag (Y or N).

Request_Exp_Type

get_null_indicator function

Purpose

Retrieve the nullable indicator.

Syntax

```
short get_null_indicator ()
```

Input arguments

None.

Output arguments

None.

Return value

Null indicator flag, which is SQL_NULLABLE or SQL_NO_NULLS.

get_data_type function

Purpose

Retrieve the data type.

Syntax

```
short get_data_type ()
```

Input arguments

None.

Output arguments

None.

Return value

Data type (SQL_TYP_XXX).

get_maximum_length function

Purpose

Retrieve the maximum length for the type.

Syntax

```
int get_maximum_length ()
```

Input arguments

None.

Output arguments

None.

Return value

Maximum length for the type.

get_precision function**Purpose**

Get the precision for numeric types.

Syntax

```
unsigned char get_precision ()
```

Input arguments

None.

Output arguments

None.

Return value

Numeric precision.

get_scale function**Purpose**

Get the scale for a numeric type.

Syntax

```
unsigned char get_scale ()
```

Input arguments

None.

Output arguments

None.

Return value

Numeric scale.

get_codepage function**Purpose**

Retrieve the code page for character types.

Syntax

```
unsigned short get_codepage ()
```

Input arguments

None.

Output arguments

None.

Request_Exp_Type

Return value

Code page.

Related reference:

- “Request classes for the C++ API” on page 149

Request_Constant class

This topic describes the Request_Constant class and provides details for each member function.

Overview

The Request_Constant class represents a data value for a Request_Exp node with a type of Request_Exp::constant.

The Request_Constant class is one of the request classes for the C++ API.

Usage This class is never instantiated by the wrapper.

File sqlqg_runtime_data_operation.h

Data members

None.

Member functions

The following table describes each member function of the Request_Constant class. Each function is described in more detail after the table.

Table 290. Member functions for the Request_Constant class

Member function	Description
get_actual_length	Retrieve the actual length of the data value.
get_data	Retrieve a pointer to the data value.
is_data_null	Indicate whether the data value is null or is not null.
get_for_bit_data	Retrieve the FOR BIT DATA flag.
get_null_indicator	Retrieve the nullable indicator.
get_data_type	Retrieve the data type.
get_maximum_length	Retrieve the maximum length for the type.
get_precision	Get the precision for numeric types.
get_scale	Get the scale for a numeric type.
get_codepage	Retrieve the code page for character types.

get_actual_length function**Purpose**

Retrieve the actual length of the data value.

Syntax

```
int get_actual_length ()
```

Input arguments

None.

Output arguments

None.

Return value

Actual data length.

get_data function**Purpose**

Retrieve a pointer to the data value.

Syntax

```
unsigned char* get_data ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the data value.

is_data_null function**Purpose**

Indicate whether the data value is null or is not null.

Syntax

```
sqlint32 is_data_null ()
```

Input arguments

None.

Output arguments

None.

Return value

Null indication (TRUE or FALSE).

Request_Constant

get_for_bit_data function

Purpose

Retrieve the FOR BIT DATA flag for the data.

Syntax

```
unsigned char get_for_bit_data ()
```

Input arguments

None.

Output arguments

None.

Return value

FOR BIT DATA flag (Y or N).

get_null_indicator function

Purpose

Retrieve the nullable indicator.

Syntax

```
short get_null_indicator ()
```

Input arguments

None.

Output arguments

None.

Return value

Null indicator flag, which is SQL_NULLABLE or SQL_NO_NULLS.

get_data_type function

Purpose

Retrieve the data type.

Syntax

```
short get_data_type ()
```

Input arguments

None.

Output arguments

None.

Return value

Data type (SQL_TYP_XXX). See the sql.h header file for SQL_TYP_XXX symbols.

get_maximum_length function**Purpose**

Retrieve the maximum length for the type.

Syntax

```
int get_maximum_length ()
```

Input arguments

None.

Output arguments

None.

Return value

Maximum length for the type.

get_precision function**Purpose**

Get the precision for numeric types.

Syntax

```
unsigned char get_precision ()
```

Input arguments

None.

Output arguments

None.

Return value

Numeric precision.

get_scale function**Purpose**

Get the scale for a numeric type.

Syntax

```
unsigned char get_scale ()
```

Input arguments

None.

Request_Constant

Output arguments

None.

Return value

Numeric scale.

get_codepage function

Purpose

Retrieve the code page for character types.

Syntax

```
unsigned short get_codepage ()
```

Input arguments

None.

Output arguments

None.

Return value

Code page.

Related reference:

- “Request classes for the C++ API” on page 149

Predicate_List class

This topic describes the Predicate_List class and provides details for each member function.

Overview

The Predicate_List class encapsulates a list of predicates and is used by the RRC protocol.

The Predicate_List class is one of the request classes for the C++ API.

Usage The wrapper does not instantiate this class unless a code for a custom cost model invokes the get_selectivity() method for a specific list of predicates.

File sqlqg_request.h

Data members

None.

Member functions

The following table describes each member function of the Predicate_List class. Each function is described in more detail after the table.

Table 291. Member functions for the Predicate_List class

Member function	Description
create_empty_predicate_list	Instantiate an empty predicate list.
create_and_copy_predicate_list	Create a predicate list that contains all of the predicates for a specific reply.
get_number_of_predicates	Retrieve the number of predicates in the list.
get_predicate_handle	Retrieve a predicate handle for a specific position in the list.
get_predicate	Retrieve a predicate expression for an expression handle.
get_number_of_applied_predicates	Get the number of predicates that are already applied.
get_applied_predicate_handle	Retrieve an applied predicate handle for a specified position in the list.
get_applied_predicate	Retrieve an applied predicate expression for an expression handle.
add_predicate	Add a predicate to the list.
add_applied_predicate	Add a predicate expression to the applied predicates list.

create_empty_predicate_list function

Purpose

Instantiate an empty predicate list.

Syntax

```
static sqlint32 create_empty_predicate_list
(Reply*          a_reply,
 Predicate_List** a_pred_list )
```

Input arguments

Table 292. Input arguments for the create_empty_predicate_list member function

Name	Data type	Description
a_reply	Reply*	Reply that owns the predicates.

Predicate_List

Output arguments

Table 293. Output arguments for the `create_empty_predicate_list` member function

Name	Data type	Description
<code>a_pred_list</code>	<code>Predicate_List**</code>	Pointer to the new predicate list.

Return value

Return code. A value of 0 indicates success.

`create_and_copy_predicate_list` function

Purpose

Create a predicate list that contains all of the predicates for a specific reply.

Syntax

```
static sqlint32 create_and_copy_predicate_list  
(Reply* a_reply,  
 Predicate_List** a_pred_list)
```

Input arguments

Table 294. Input arguments for the `create_and_copy_predicate_list` member function

Name	Data type	Description
<code>a_reply</code>	<code>Reply*</code>	Reply that owns the predicates.

Output arguments

Table 295. Output arguments for the `create_and_copy_predicate_list` member function

Name	Data type	Description
<code>a_pred_list</code>	<code>Predicate_List**</code>	Pointer to the new predicate list.

Return value

Return code. A value of 0 indicates success.

`get_number_of_predicates` function

Purpose

Retrieve the number of predicates in the list.

Syntax

```
int get_number_of_predicates ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of predicates.

get_predicate_handle function

Purpose

Retrieve a predicate handle for a specific position in the list.

Syntax

```
sqlint32 get_predicate_handle (int a_pred_pos,
                               int* a_pred_handle)
```

Input arguments

Table 296. Input arguments for the get_predicate_handle member function

Name	Data type	Description
a_pred_pos	int	Position of a predicate in the list (starting at 1).

Output arguments

Table 297. Output arguments for the get_predicate_handle member function

Name	Data type	Description
a_pred_handle	int*	Predicate expression handle.

Return value

Return code. A value of 0 indicates success.

get_predicate function

Purpose

Retrieve a predicate expression for an expression handle.

Syntax

```
sqlint32 get_predicate (int a_pred_handle,
                       Request_Exp** a_pred_exp)
```

Predicate_List

Input arguments

Table 298. Input arguments for the `get_predicate` member function

Name	Data type	Description
<code>a_pred_handle</code>	<code>int</code>	Predicate expression handle.

Output arguments

Table 299. Output arguments for the `get_predicate` member function

Name	Data type	Description
<code>a_pred_exp</code>	<code>Request_Exp**</code>	Pointer to an expression node.

Return value

Return code. A value of 0 indicates success.

`get_number_of_applied_predicates` function

Purpose

Get the number of predicates that are already applied.

Syntax

```
int get_number_of_applied_predicates ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of applied predicates.

`get_applied_predicate_handle` function

Purpose

Retrieve an applied predicate handle for a specified position in the list.

Syntax

```
sqlint32 get_applied_predicate_handle (int a_applied_pred_pos,  
int* a_applied_pred_handle)
```


Input arguments

Table 300. Input arguments for the `get_applied_predicate_handle` member function

Name	Data type	Description
<code>a_applied_pred_pos</code>	<code>int</code>	Position of the predicate in the list (starting at 1).

Output arguments

Table 301. Output arguments for the `get_applied_predicate_handle` member function

Name	Data type	Description
<code>a_applied_pred_handle</code>	<code>int*</code>	Applied predicate expression handle.

Return value

Return code. A value of 0 indicates success.

get_applied_predicate function

Purpose

Retrieve an applied predicate expression for an expression handle.

Syntax

```
sqlint32 get_applied_predicate (int a_applied_pred_handle,
                                Request_Exp** a_applied_pred_exp)
```

Input arguments

Table 302. Input arguments for the `get_applied_predicate` member function

Name	Data type	Description
<code>a_applied_pred_handle</code>	<code>int</code>	Applied predicate expression handle.

Output arguments

Table 303. Output arguments for the `get_applied_predicate` member function

Name	Data type	Description
<code>a_applied_pred_exp</code>	<code>Request_Exp**</code>	Pointer to an expression node.

Return value

Return code. A value of 0 indicates success.

Predicate_List

add_predicate function

Purpose

Add a predicate to the list.

Syntax

```
sqlint32 add_predicate (int a_pred_handle)
```

Input arguments

Table 304. Input arguments for the add_predicate member function

Name	Data type	Description
a_pred_handle	int	Handle for the predicate expression.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

add_applied_predicate function

Purpose

Add a predicate expression to the applied predicates list.

Syntax

```
sqlint32 add_applied_predicate (int a_applied_pred_handle)
```

Input arguments

Table 305. Input arguments for the add_applied_predicate member function

Name	Data type	Description
a_applied_pred_handle	int	Handle for the applied predicate expression.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Request classes for the C++ API” on page 149

Data classes for the C++ API

The following table describes each data class for the C++ API.

Table 306. Data classes

Class name	Description
Runtime_Data_Desc	The class that encapsulates the description of a data item.
Runtime_Data_Desc_List	A list of Runtime_Data_Desc.
Runtime_Data	The class that encapsulates a data value that is passed between the wrapper and the query gateway interface. This data value can be column data that is in a result or is in a query parameter.
Runtime_Data_List	A list of pointers to Runtime_Data objects.

Related reference:

- “Runtime_Data_Desc class” on page 195
- “Runtime_Data_Desc_List class” on page 200
- “Runtime_Data class” on page 203
- “Runtime_Data_List class” on page 212

Runtime_Data_Desc class

This topic describes the Runtime_Data_Desc class and provides details for each member function.

Overview

The Runtime_Data_Desc class encapsulates the description of a data item.

The Runtime_Data_Desc class is one of the data classes for the C++ API.

Usage This class is instantiated by the DB2 query gateway and is never subclassed by the wrapper. This class can be instantiated by the wrapper during the prepare processing for a pass-through operation.

File sqlqg_runtime_data_operation.h

Data members

None.

Constructors and member functions

The following tables describe the constructor and the member functions of the Runtime_Data_Desc class. The constructor and functions are described in more detail after the tables.

Runtime_Data_Desc

Table 307. Constructors for the Runtime_Data_Desc class

Constructor	Description
Runtime_Data_Desc	Construct an instance of Runtime_Data_Desc.

Table 308. Member functions for the Runtime_Data_Desc class

Member function	Description
get_for_bit_data	Retrieve the FOR BIT DATA flag for the data.
get_null_indicator	Retrieve the nullable indicator.
get_data_type	Retrieve the data type.
get_maximum_length	Retrieve the maximum length for the type.
get_precision	Get the precision for numeric types.
get_scale	Get the scale for a numeric type.
get_codepage	Retrieve the code page for character types.

Runtime_Data_Desc constructor

Purpose

Construct an instance of Runtime_Data_Desc. This constructor is used when instantiating the Runtime_Data_Desc class in response to a prepare or describe operation for a pass-through session.

Syntax

```
Runtime_Data_Desc (sqlint32* a_rc,  
                  short a_type,  
                  int a_max_length,  
                  short a_codepage,  
                  short a_null_ind,  
                  unsigned char a_data_precision=0,  
                  unsigned char a_data_scale=0,  
                  sqluint8* a_data_name=NULL,  
                  short a_name_length=0,  
                  short a_remote_type=0)
```

Input arguments

Table 309. Input arguments for the Runtime_Data_Desc constructor

Name	Data type	Description
a_type	short	Data type. Use the SQL_TYP_XXX values that are defined in the sql.h header file.

Table 309. Input arguments for the Runtime_Data_Desc constructor (continued)

Name	Data type	Description
a_max_length	int	Maximum length of the data.
a_codepage	short	Code page for the character data.
a_null_ind	short	Null indicator.
a_data_precision	unsigned char	Precision for numeric data.
a_data_scale	unsigned char	Scale for a numeric data type.
a_data_name	sqluint8*	Name of the data item.
a_name_length	short	Length of the name.
a_remote_type	short	Remote type code.

Output arguments

Table 310. Output arguments for the Runtime_Data_Desc constructor

Name	Data type	Description
a_rc	sqlint32*	Pointer to the return code value; 0 indicates success.

get_for_bit_data function**Purpose**

Retrieve the FOR BIT DATA flag for the data.

Syntax

```
unsigned char get_for_bit_data ()
```

Input arguments

None.

Output arguments

None.

Return value

FOR BIT DATA flag (Y or N).

get_null_indicator function**Purpose**

Retrieve the nullable indicator.

Runtime_Data_Desc

Syntax

```
short get_null_indicator ()
```

Input arguments

None.

Output arguments

None.

Return value

Null indicator flag, which is `SQL_NULLABLE` or `SQL_NO_NULLS`.

get_data_type function

Purpose

Retrieve the data type.

Syntax

```
short get_data_type ()
```

Input arguments

None.

Output arguments

None.

Return value

Data type (`SQL_TYP_XXX`). See the `sql.h` header file for the `SQL_TYP_XXX` symbols.

get_maximum_length function

Purpose

Retrieve the maximum length for the type.

Syntax

```
int get_maximum_length ()
```

Input arguments

None.

Output arguments

None.

Return value

Maximum length for the type.

get_precision function**Purpose**

Get the precision for numeric types.

Syntax

```
unsigned char get_precision ()
```

Input arguments

None.

Output arguments

None.

Return value

Numeric precision.

get_scale function**Purpose**

Get the scale for a numeric type.

Syntax

```
unsigned char get_scale ()
```

Input arguments

None.

Output arguments

None.

Return value

Numeric scale.

get_codepage function**Purpose**

Retrieve the code page for character types.

Syntax

```
unsigned short get_codepage ()
```

Input arguments

None.

Output arguments

None.

Return value

Code page.

Runtime_Data_Desc

Related reference:

- “Data classes for the C++ API” on page 195

Runtime_Data_Desc_List class

This topic describes the Runtime_Data_Desc_List class and provides details for each member function.

Overview

The Runtime_Data_Desc_List class provides a list of Runtime_Data_Desc class.

The Runtime_Data_Desc_List class is one of the data classes for the C++ API.

Usage This class is instantiated by the DB2 query gateway but is never subclassed by the wrapper.

File sqlqg_runtime_data_operation.h

Data members

None.

Member functions

The following table describes each member function of the Runtime_Data_Desc_List class. Each function is described in more detail after the table.

Table 311. Member functions for the Runtime_Data_Desc_List class

Member function	Description
get_number_of_values	Retrieve the number of values.
set_number_of_values	Set the number of values in the list.
get_ith_value	Retrieve the <i>ith</i> data descriptor.
set_ith_value	Store a Runtime_Data_Desc pointer in the <i>ith</i> position of the list.
operator[]	Retrieve the <i>ith</i> entry in the list by using subscript notation.

get_number_of_values function

Purpose

Retrieve the number of values.

Syntax

```
int get_number_of_values ()
```


Input arguments

None.

Output arguments

None.

Return value

Number of values.

set_number_of_values function**Purpose**

Set the number of values in the list. This member function can empty the list by setting the number to 0. This member function can also lengthen the list or shorten the list.

Syntax

```
sqlint32 set_number_of_values (int a_count)
```

Input arguments

Table 312. Input arguments for the set_number_of_values member function

Name	Data type	Description
a_count	int	Number of values.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_ith_value function**Purpose**

Retrieve the *ith* data descriptor.

Syntax

```
Runtime_Data_Desc* get_ith_value (int a_index)
```

Input arguments

Table 313. Input arguments for the get_ith_value member function

Name	Data type	Description
a_index	int	Index into the list (starting at 0).

Runtime_Data_Desc_List

Output arguments

None.

Return value

Pointer to the *i*th value (or null).

set_ith_value function

Purpose

Store a Runtime_Data_Desc pointer in the *i*th position of the list.

Syntax

```
sqlint32 set_ith_value (Runtime_Data_Desc* a_desc,  
                       int a_index)
```

Input arguments

Table 314. Input arguments for the set_ith_value member function

Name	Data type	Description
a_desc	Runtime_Data_Desc*	Descriptor pointer.
a_index	int	Index (starting at 0).

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

operator[] function

Purpose

Retrieve the *i*th entry in the list by using subscript notation.

Syntax

```
Runtime_Data_Desc* operator[] (int a_index)
```

Input arguments

Table 315. Input arguments for the operator [] member function

Name	Data type	Description
a_index	int	Index into the list (starting at 0).

Output arguments

None.

Return value

The *i*th entry.

Related reference:

- “Data classes for the C++ API” on page 195

Runtime_Data class

This topic describes the Runtime_Data class and provides details for each member function.

Overview

The Runtime_Data class encapsulates a data value that is passed between the wrapper and the query gateway interface. This data value can be column data that is in a result or is in a query parameter.

The Runtime_Data class is one of the data classes for the C++ API.

Usage This class is instantiated by the DB2 query gateway and is never subclassed by the wrapper.

File sqlqg_runtime_data_operation.h

Data members

None.

Member functions

The following table describes each member function of the Runtime_Data class. Each function is described in more detail after the table.

Table 316. Member functions for the Runtime_Data class

Member function	Description
get_actual_length	Retrieve the actual length of the data value.
set_actual_length	Set the actual length of the data value.
get_data	Retrieve a pointer to the data value.
set_data	Copy information into the data value.
is_data_null	Indicate whether the data value is or is not null.
set_data_null	Mark the data value as null.
clear_null_indicator	Reset the null indicator for the data value.
set_friendly_div_by_0	Indicate that a value is null because a divide by zero error occurred.

Table 316. Member functions for the Runtime_Data class (continued)

Member function	Description
set_friendly_exception	Indicate that a value is null because of a numeric exception.
is_data_nullable	Retrieve an indication that the data value is or is not nullable.
is_semantic_null	Return an indication that the value is semantically null.
check_friendly_div_by_0	Determine if the reason for a null indication is a divide by zero error.
check_friendly_exception	Determine if the reason for a null indication is a numeric exception.
get_for_bit_data	Retrieve the FOR BIT DATA flag for the data.
get_null_indicator	Retrieve the nullable indicator.
get_data_type	Retrieve the data type.
get_maximum_length	Retrieve the maximum length.
get_precision	Get the precision for numeric types.
get_scale	Get the scale for a numeric type.
get_codepage	Retrieve the code page for character types.

get_actual_length function

Purpose

Retrieve the actual length of the data value.

Syntax

```
int get_actual_length ()
```

Input arguments

None.

Output arguments

None.

Return value

Actual data length.

set_actual_length function

Purpose

Set the actual length of the data value.

Syntax

```
void set_actual_length (int a_length)
```

Input arguments

Table 317. Input arguments for the set_actual_length member function

Name	Data type	Description
a_length	int	Length.

Output arguments

None.

Return value

None.

get_data function**Purpose**

Retrieve a pointer to the data value.

Syntax

```
unsigned char* get_data ()
```

Input arguments

None.

Output arguments

None.

Return value

Pointer to the data value.

set_data function**Purpose**

Copy information into the data value.

Syntax

```
sqlint32 set_data (unsigned char* a_data_ptr,
                  int a_copy_len)
```

Input arguments

Table 318. Input arguments for the set_data member function

Name	Data type	Description
a_data_ptr	unsigned char*	Data value.

Table 318. Input arguments for the `set_data` member function (continued)

Name	Data type	Description
<code>a_copy_len</code>	<code>int</code>	Length of the value.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

`is_data_null` function

Purpose

Indicate whether the data value is or is not null.

Syntax

```
sqlint32 is_data_null ()
```

Input arguments

None.

Output arguments

None.

Return value

Null indication.

`set_data_null` function

Purpose

Mark the data value as null.

Syntax

```
sqlint32 set_data_null ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

clear_null_indicator function**Purpose**

Reset the null indicator for the data value.

Syntax

```
sqlint32 clear_null_indicator ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

set_friendly_div_by_0 function**Purpose**

Indicate that a value is null because a divide by zero error occurred.

Syntax

```
sqlint32 set_friendly_div_by_0 ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

set_friendly_exception function**Purpose**

Indicate that a value is null because of a numeric exception.

Syntax

```
sqlint32 set_friendly_exception ()
```

Input arguments

None.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

is_data_nullable function

Purpose

Retrieve an indication that the data value is or is not nullable.

Syntax

```
sqlint32 is_data_nullable ()
```

Input arguments

None.

Output arguments

None.

Return value

Nullable indication. The value is TRUE if nullable.

is_semantic_null function

Purpose

Return an indication that the value is semantically null.

Syntax

```
sqlint32 is_semantic_null ()
```

Input arguments

None.

Output arguments

None.

Return value

Nullable indication. The value is TRUE if nullable.

check_friendly_div_by_0 function

Purpose

Determine if the reason for a null indication is a divide by zero error.

Syntax

```
sqlint32 check_friendly_div_by_0
```

Input arguments

None.

Output arguments

None.

Return value

Null indication. The value is TRUE if a divide by zero condition occurred.

check_friendly_exception function**Purpose**

Determine if the reason for a null indication is a numeric exception.

Syntax

```
sqlint32 check_friendly_exception ()
```

Input arguments

None.

Output arguments

None.

Return value

Null indication. The value is TRUE if an exception occurred.

get_for_bit_data function**Purpose**

Retrieve the FOR BIT DATA flag for the data.

Syntax

```
unsigned char get_for_bit_data ()
```

Input arguments

None.

Output arguments

None.

Return value

FOR BIT DATA flag (Y or N).

get_null_indicator function**Purpose**

Retrieve the nullable indicator.

Syntax

```
short get_null_indicator ()
```

Input arguments

None.

Runtime_Data

Output arguments

None.

Return value

Null indicator flag, which is `SQL_NULLABLE` or `SQL_NO_NULLS`.

get_data_type function

Purpose

Retrieve the data type.

Syntax

```
short get_data_type ()
```

Input arguments

None.

Output arguments

None.

Return value

Data type (`SQL_TYP_xxx`). See the `sql.h` header file for the `SQL_TYP_xxx` symbols.

get_maximum_length function

Purpose

Retrieve the maximum length for the type.

Syntax

```
int get_maximum_length ()
```

Input arguments

None.

Output arguments

None.

Return value

Maximum length for the type.

get_precision function

Purpose

Get the precision for numeric types.

Syntax

```
unsigned char get_precision ()
```

Input arguments

None.

Output arguments

None.

Return value

Numeric precision.

get_scale function**Purpose**

Get the scale for a numeric type.

Syntax

```
unsigned char get_scale ()
```

Input arguments

None.

Output arguments

None.

Return value

Numeric scale.

get_codepage function**Purpose**

Retrieve the code page for character types.

Syntax

```
unsigned short get_codepage ()
```

Input arguments

None.

Output arguments

None.

Return value

Code page.

Related reference:

- “Data classes for the C++ API” on page 195

Runtime_Data_List class

This topic describes the Runtime_Data_List class and provides details for each member function.

Overview

The Runtime_Data_List class provides a list of pointers to Runtime_Data objects.

The Runtime_Data_List class is one of the data classes for the C++ API.

Usage This class is instantiated by the DB2 query gateway and is never subclassed by the wrapper.

File sqlqg_runtime_data_operation.h

Data members

None.

Member functions

The following table describes each member function of the Runtime_Data_List class. Each function is described in more detail after the table.

Table 319. Member functions for the Runtime_Data_List class

Member function	Description
get_number_of_values	Retrieve the number of values.
get_ith_value	Retrieve the <i>ith</i> data descriptor.
operator[]	Retrieve the <i>ith</i> entry in the list by using subscript notation.

get_number_of_values function

Purpose

Retrieve the number of values.

Syntax

```
int get_number_of_values ()
```

Input arguments

None.

Output arguments

None.

Return value

Number of values.

get_ith_value function**Purpose**

Retrieve the *i*th data value.

Syntax

```
Runtime_Data* get_ith_value (int a_index)
```

Input arguments

Table 320. Input arguments for the get_ith_value member function

Name	Data type	Description
a_index	int	Index into the list (starting at 0).

Output arguments

None.

Return value

Pointer to the *i*th value (or null).

operator[] function**Purpose**

Retrieve the *i*th entry in the list by using subscript notation.

Syntax

```
Runtime_Data* operator[] (int a_index)
```

Input arguments

Table 321. Input arguments for the operator[] member function

Name	Data type	Description
a_index	int	Index into the list (starting at 0).

Output arguments

None.

Return value

The *i*th entry.

Related reference:

- “Data classes for the C++ API” on page 195

Wrapper_Utilities class

This topic describes the Wrapper_Utilities class and provides details for each member function.

Overview

The Wrapper_Utilities class is a container for several static utility functions. Do not instantiate or subclass the Wrapper_Utilities class.

The Wrapper_Utilities class is a utilities class for the C++ API.

File sqlqg_utils.h

Data members

None.

Member functions

The following table describes each member function of the Wrapper_Utilities class. Each function is described in more detail after the table.

Table 322. Member functions for the Wrapper_Utilities class

Member function	Description
convert_to_upper	Convert a character string to uppercase by using the specified code page.
convert_to_lower	Convert a character string to lowercase by using the specified code page.
report_error	Generate an error to report to the user.
report_warning	Generate a warning to report to the user.
allocate	Allocate a block of memory.
deallocate	Free a block of memory that is allocated with allocate().
get_sb_DB_codepage	Return the single-byte code page for the current database.
get_db_DB_codepage	Return the double-byte code page for the current database.
string_to_tokens	Scan a character string and divide the string into successive tokens.
get_db2_install_path	Return a null-terminated character string that shows the absolute path name of the DB2 Information Integrator installation directory.
get_db2_instance_path	Return a null-terminated character string that shows the absolute path name of the DB2 Information Integrator instance.

Table 322. Member functions for the Wrapper_Utilities class (continued)

Member function	Description
trace_data	Write a block of information to the DB2 trace facility.

convert_to_upper function**Purpose**

Convert a character string to uppercase by using the specified code page.

Usage Use the get_sb_DB_codepage() or the get_db_DB_codepage() function to obtain the current database code page.

Syntax

```
int convert_to_upper (char*      a_string,
                    size_t     a_length,
                    unsigned int a_codepage)
```

Input arguments

Table 323. Input arguments for the convert_to_upper member function

Name	Data type	Description
a_string	char*	String to convert (not null-terminated).
a_length	size_t	Length of the string.
a_codepage	unsigned int	Code page for conversion.

Output arguments

None.

Return value

Return code. A value of 0 indicates success. A value of -1 indicates that the code page is undefined and that the string to be converted contains extended characters. These extended characters are skipped.

convert_to_lower function**Purpose**

Convert a character string to lowercase by using the specified code page.

Usage Use the get_sb_DB_codepage() or the get_db_DB_codepage() function to obtain the current database code page.

Syntax

```
int convert_to_lower (char*      a_string,  
                    size_t     a_length,  
                    unsigned int a_codepage)
```

Input arguments

Table 324. Input arguments for the `convert_to_lower` member function

Name	Data type	Description
<code>a_string</code>	<code>char*</code>	String to convert (not null-terminated).
<code>a_length</code>	<code>size_t</code>	Length of the string.
<code>a_codepage</code>	<code>unsigned int</code>	Code page for conversion.

Output arguments

None.

Return value

Return code. A value of 0 indicates success. A value of -1 indicates that the code page is undefined and that the string to be converted contains extended characters. These extended characters are skipped.

report_error function

Purpose

Generate an error to report to the user.

Syntax

```
sqlint32 report_error (const char* a_function_name,  
                    int           a_sql_code,  
                    int           a_number_of_tokens,  
                    ...)
```

Input arguments

Table 325. Input arguments for the `report_error` member function

Name	Data type	Description
<code>a_function_name</code>	<code>const char*</code>	Null-terminated character string that identifies the function that generates the error. The string cannot be greater than five characters. This string value is accessible to the client program through the <code>sqlerrp</code> field of the <code>SQLCA</code> and appears in uppercase letters with a prefix of <code>SQL</code> .

Table 325. Input arguments for the `report_error` member function (continued)

Name	Data type	Description
<code>a_sql_code</code>	<code>int</code>	The predefined SQL code for the error. See the <code>sqlcodes.h</code> file.
<code>a_number_of_tokens</code>	<code>int</code>	The number of substitution tokens for the message.
...	<code>(int, const char*)</code>	One pair of values for each token. Each pair consists of an integer length and a pointer to a character string that is not null-terminated.

Output arguments

None.

Return value

Return code. This function returns the reported error code to the caller, and the caller returns this error code to DB2.

report_warning function**Purpose**

Generate a warning to report to the user.

Syntax

```
sqlint32 report_warning (const char* a_function_name,
                        int          a_warning_index,
                        char         a_warning_flag,
                        int          a_number_of_tokens,
                        ...)
```

Input argumentsTable 326. Input arguments for the `report_warning` member function

Name	Data type	Description
<code>a_function_name</code>	<code>const char*</code>	Null-terminated character string that identifies the function that generates the warning. The string cannot be greater than five characters. This string value is accessible to the client program through the <code>sqlerrp</code> field of the <code>SQLCA</code> and appears in uppercase letters with a prefix of <code>SQL</code> .

Table 326. Input arguments for the `report_warning` member function (continued)

Name	Data type	Description
<code>a_warning_index</code>	<code>int</code>	Value that identifies the type of warning. Use the <code>SQL_WARN_XXX</code> constants that are defined in the <code>sql.h</code> header file.
<code>a_warning_flag</code>	<code>char</code>	Warning flag. Use the <code>SQL_WARNING</code> constant that is defined in the <code>sql.h</code> header file.
<code>a_number_of_tokens</code>	<code>int</code>	Number of tokens (length and string pairs) to be supplied through the <code>sqlerrm</code> field of the <code>SQLCA</code> and is accessible to a client program.
...	(<code>int</code> , <code>const char*</code>)	One pair of values for each token. Each pair consists of an integer length and a pointer to a character string that is not null-terminated.

Output arguments

None.

Return value

A value of 0 indicates that the warning generated successfully. A non-zero value indicates that a problem occurred when the warning generated. A non-zero return code must be returned to DB2.

allocate function

Purpose

Allocate a block of memory.

Syntax

```
int allocate (size_t a_size,
             void** a_block)
```

Input arguments

Table 327. Input arguments for the `allocate` member function

Name	Data type	Description
<code>a_size</code>	<code>size_t</code>	Size of a block to allocate.

Output arguments*Table 328. Output arguments for the allocate member function*

Name	Data type	Description
a_block	void**	Pointer to the allocated block.

Return value

Return code. A value of 0 indicates success.

deallocate function**Purpose**

Free a block of memory that is allocated with allocate().

Syntax

```
void deallocate (void* a_block)
```

Input arguments*Table 329. Input arguments for the deallocate member function*

Name	Data type	Description
a_block	void*	Block of memory to free.

Output arguments

None.

Return value

None.

get_sb_DB_codepage function**Purpose**

Return the single-byte code page for the current database.

Syntax

```
int get_sb_DB_codepage ()
```

Input arguments

None.

Output arguments

None.

Return value

Code page.

get_db_DB_codepage function

Purpose

Return the double-byte code page for the current database.

Syntax

```
int get_db_DB_codepage ()
```

Input arguments

None.

Output arguments

None.

Return value

Code page.

string_to_tokens function

Purpose

Scan a character string and divide the string into successive tokens. This member function is a safe alternative to the strtok() and strtok_r() functions.

Syntax

```
char* string_to_tokens (char*      a_string,  
                      const char* a_sep,  
                      char**     a_last)
```

Input arguments

Table 330. Input arguments for the string_to_tokens member function

Name	Data type	Description
a_string	char*	String to be scanned.
a_sep	const char*	String to be used as a separator between the tokens.
a_last	char*	Value that is used to retain the state between the calls to string_to_tokens.

Output arguments

None.

Return value

Pointer to the next token in string or null.

get_db2_install_path function**Purpose**

Return a null-terminated character string that shows the absolute path name of the DB2 Information Integrator installation directory.

Syntax

```
sqlint32 get_db2_install_path (char*   a_path,
                              sqlint32 a_path_size)
```

Input arguments

Table 331. Input arguments for the get_db2_install_path member function

Name	Data type	Description
a_path	char*	Pointer to the buffer that contains the path name.
a_path_size	sqlint32	Length of the buffer (including space for a null-terminator).

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

get_db2_instance_path function**Purpose**

Return a null-terminated character string that shows the absolute path name of the DB2 Information Integrator instance.

Syntax

```
sqlint32 get_db2_instance_path (char*   a_path,
                                sqlint32 a_path_size)
```

Input arguments

Table 332. Input arguments for the get_db2_instance_path member function

Name	Data type	Description
a_path	char*	Pointer to the buffer that contains the path name.
a_path_size	sqlint32	Length of the buffer (including space for a null-terminator).

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

trace_data function

Purpose

Write a block of information to the DB2 trace facility.

Usage For more information about the DB2 trace facility, see *DB2 Command Reference*.

Syntax

```
void trace_data (int    a_probe,  
                void*  a_data,  
                int    a_data_size)
```

Input arguments

Table 333. Input arguments for the trace_data member function

Name	Data type	Description
a_probe	int	Number that identifies the trace point.
a_data	void*	Pointer to the data to be traced.
a_data_size	int	Length of the data to be traced.

Output arguments

None.

Return value

Return code. A value of 0 indicates success.

Related reference:

- “Catalog classes for the C++ API” on page 1
- “Data classes for the C++ API” on page 195

DB2 Information Integrator technical documentation

The following topics describe how to:

- Access books and release information, including printing and ordering books
- Access topics by using the DB2 Information Integrator Information Center or the DB2 HTML Documentation CD

Accessing books and release information

DB2 Information Integrator technical information is available in the following formats:

- Books (PDF and printed). A description of each book in the DB2 Information Integrator library is available from the IBM Publications Center at www.ibm.com/shop/publications/order.
- An information center (HTML format).
- Help for DB2 database tools (HTML format).

DB2 Information Integrator books

The DB2 Information Integrator PDF Documentation CD contains PDF files of the books in the DB2 Information Integrator library and the DB2 Universal Database library. The structure of the DB2 Information Integrator PDF Documentation CD is:

- On Windows operating systems: *x:\doc\%L*
- On UNIX operating systems: */cdrom/doc/%L/*

where:

- *x* represents the Windows CD-ROM drive letter
- *cdrom* refers to the UNIX mount point of the CD-ROM
- *%L* is the locale of the documentation that you want to use, for example, *en_US*

Language	Locale	Identifier	Language	Locale	Identifier
Arabic	ar_AA	w	Japanese	ja_JP	j
Brazilian Portuguese	pt_BR	b	Korean	ko_KR	k
Bulgarian	bg_BG	u	Norwegian	no_NO	n
Croatian	hr_HR	9	Polish	pl_PL	p
Czech	cs_CZ	x	Portuguese	pt_PT	v
Danish	da_DK	d	Romanian	ro_RO	8
Dutch	nl_NL	q	Russian	ru_RU	r
English	en_US	e	Simplified Chinese	zh_CN	c

Finnish	fi_FI	y	Slovakian	sk_SK	7
French	fr_FR	f	Slovenian	sl_SI	l
German	de_DE	g	Spanish	es_ES	z
Greek	el_GR	a	Swedish	sv_SE	s
Hungarian	hu_HU	h	Traditional Chinese	zh_TW	t
Italian	it_IT	i	Turkish	tr_TR	m

The character in the sixth position of each PDF file name indicates the language version of a book (see the following table). For example, the file name `iiyige80` identifies the English version of the *IBM DB2 Information Integrator Installation Guide*, and the file name `iiyigg80` identifies the German version of the same book.

The books in the following table are available for DB2 Information Integrator.

Table 334. DB2 Information Integrator documentation

Name	Form number	Install category	PDF file name
<i>IBM DB2 Information Integrator Solutions Guide</i>	SC18-7037	getting_started	iiyisx80
<i>IBM DB2 Information Integrator Installation Guide</i>	GC18-7036	getting_started	iiyigx80
<i>IBM DB2 Information Integrator Migration Guide</i>	SC18-7360	getting_started	iiymgx80
<i>IBM DB2 Information Integrator Federated Systems Guide</i>	SC18-7364	admin	iiyfp80
<i>IBM DB2 Information Integrator Data Source Configuration Guide</i>	Available online only	optional	iiylsx80
<i>IBM DB2 Information Integrator Developer's Guide</i>	SC18-7359	ad	iiyfsx80

Printing books from PDF files

You can print DB2 Information Integrator books from the PDF files on the DB2 Information Integrator PDF Documentation CD. You can use Adobe Acrobat Reader to print the entire book, a range of pages, or specific pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com.

Procedure:

To print a DB2 Information Integrator book from a PDF file:

1. Insert the DB2 Information Integrator PDF Documentation CD. On UNIX operating systems, mount the CD.
2. Start the Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
 - On Windows operating systems: `x:\doc\%L`
 - On UNIX operating systems: `/cdrom/doc/%L/`where:
 - `x` represents the Windows CD-ROM drive letter
 - `cdrom` refers to the UNIX mount point of the CD-ROM
 - `%L` is the locale of the documentation that you want to print, for example, `en_US`
4. Click **File** → **Print**.
5. In the Print window, specify whether to print all of the pages, the current pages, or a range of pages.
6. Click **OK**.

Ordering printed books

You can get printed manuals by ordering the documentation package (doc pack) for your DB2 Information Integrator product from your IBM reseller. The doc packs are a subset of the manuals from the DB2 Information Integrator library. These doc packs are designed to help you to get started using the DB2 Information Integrator product that you purchased.

The manuals in the doc packs are the same as those on the DB2 Information Integrator PDF Documentation CD that was provided with your DB2 Information Integrator product.

You can also use one of the following methods to order individual books:

- Contact your IBM marketing representative or authorized dealer. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/planetwide.
- Phone 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.
- Go to the IBM Publications Center at www.ibm.com/shop/publications/order.

Release notes and installation requirements

The release notes and installation requirements provide information that is specific to the release and fix pack level of your product. The release notes also provide summaries of the documentation updates that are incorporated in each release and fix pack.

The release notes and the installation requirements are available in text and HTML formats on the product CD-ROM:

- On Windows operating systems: `x:\doc\%L`
- On UNIX operating systems: `/cdrom/doc/%L/`

where:

- `x` represents the Windows CD-ROM drive letter
- `cdrom` refers to the UNIX mount point of the CD-ROM
- `%L` is the locale of the documentation that you want to use; for example, `en_US`

Table 335. Release notes

Name	File name	Location
<i>DB2 Information Integrator Release Notes</i>	ReleaseNotes	<ul style="list-style-type: none">• The product CD-ROM• The DB2 Information Integrator Information Center• The DB2 Information Integration Installation Launchpad
<i>DB2 Information Integrator Installation Requirements</i>	Prereqs	<ul style="list-style-type: none">• The product CD-ROM• The DB2 Information Integration Installation Launchpad

Fix packs for DB2 Information Integrator documentation

IBM might periodically make documentation fix packs available. You can use documentation fix packs to update the information that you installed from the DB2 HTML Documentation CD when new information becomes available.

Documentation fix packs are cumulative. For example, if you install the documentation for Version 8.1 and then apply Version 8.1.2, you will get documentation updates for FixPak 1 and Version 8.1.2.

When you install documentation fix packs, your HTML documentation will be more current than the printed and online PDF manuals for your product.

Accessing topics using the DB2 Information Integrator Information Center or the DB2 HTML Documentation CD

The DB2 Information Integrator Information Center gives you access to the information that you need to take advantage of DB2 Information Integrator in your business.

Features of the DB2 Information Integrator Information Center

The DB2 Information Integrator Information Center has the following features:

Integrated navigation tree

Locate any topic in the DB2 Information Integrator library from a single navigation tree.

Search

Search all of the topics on your workstation by clicking **Search** in the navigation toolbar.

Master index

Access the information in topics and tools help from one master index. The index contains entries from the entire DB2 Information Integrator library.

Master glossary

The master glossary defines terms that are used in the DB2 Information Integrator library.

Regularly updated documentation

Keep your topics up to date by downloading updated HTML topics.

Finding topics in the DB2 Information Integrator Information Center

The following major elements comprise the DB2 Information Integrator Information Center:

Navigation tree

The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topic links, the glossary, and the master index in the DB2 Information Integrator Information Center.

Navigation toolbar

The navigation toolbar is located in the top right frame of the browser window. Use the push buttons in the navigation toolbar to search the DB2 Information Integrator Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame

The content frame is located in the bottom right frame of the browser window. When you click a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index, the content frame displays the appropriate topic.

Prerequisites:

To access the DB2 Information Integrator Information Center from a browser, you must use one of the following browsers:

- Microsoft Explorer Version 5 or later
- Netscape Navigator Version 6.1 or later

Restrictions:

The DB2 Information Integrator Information Center contains only the sets of topics that you installed from the DB2 HTML Documentation CD. If your Web browser returns a File not found error when you follow a link to a topic, install one or more additional sets of topics from the DB2 HTML Documentation CD.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest, then click **Search**. The **Results** field displays a list of topics. The topics that most closely match your search string are at the top of the list.

Enter more terms to increase the precision of your query and reduce the number of topics that are returned from your query.

3. In the **Results** field, click the title of the topic that you want to read. The topic is displayed in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon next to the category of topics in which you are interested. A list of subcategories is displayed under the icon.
2. Continue to click the book icons until you find the category that contains the topics in which you are interested. Categories that link to topics show the category title as a link when you move the cursor over the category title. A page icon is used in the navigation tree to identify topics.
3. Click the topic link. The topic is displayed in the content frame.

To find a topic by using the master index:

1. In the navigation tree, click **Index**. The index expands to display a list of links arranged in alphabetical order.

2. In the navigation tree, click the first character of the subject that you are looking for. A list of entries with that initial character is displayed in the content frame. If a book icon is displayed, the subject has multiple index entries.
3. Click the book icon that corresponds to the subject in which you are interested. A list of topics is displayed below the term that you clicked.
4. Click on the title of the topic that meets your needs. The topic is displayed in the content frame.

Using the DB2 HTML documentation

This topic describes how to install, view, and copy the documentation on the DB2 HTML Documentation CD, and how to update the documentation after you install it.

Installing the DB2 HTML documentation

The installation directory for the DB2 HTML Documentation CD differs for each category of information:

htmlcdpath/doc/htmlcd/%L/category

htmlcdpath

The directory where the DB2 HTML Documentation CD is installed.

%L The locale of the documentation that you want to use, for example, en_US.

category

The category identifier, for example, getting_started for the installation information.

Viewing technical documentation directly from the DB2 HTML Documentation CD

You can view all of the HTML topics from the DB2 HTML Documentation CD.

Restrictions:

You must install the DB2 product to view the online help.

Procedure:

To view the HTML documentation from the DB2 HTML Documentation CD:

1. Insert the DB2 HTML Documentation CD.
2. Start your Web browser and open the following file:
 - For Windows operating systems:
x:\Program Files\IBM\sqllib\doc\htmlcd\%L\index.*
 - For UNIX operating systems:
/cdrom/Program Files/IBM/sqllib/doc/htmlcd/%L/index.htm

where:

- *x* represents the Windows CD-ROM drive letter
- *cdrom* refers to UNIX mount point of the CD-ROM
- *%L* is the locale of the documentation that you want to use, for example, en_US

Copying files from the DB2 HTML Documentation CD to a Web Server

The entire DB2 library is available on the DB2 HTML Documentation CD, so you can copy the library on a Web server for easier access.

Procedure:

Copy files from the DB2 HTML Documentation CD to the appropriate path on your Web server (the default path is shown):

- For Windows operating systems: *x:\Program Files\IBM\sqliib\doc\htmlcd\%L*.**
- For UNIX operating systems: */cdrom/Program Files/IBM/sqliib/doc/htmlcd/%L*

where:

- *x* represents the Windows CD-ROM drive letter
- *cdrom* refers to UNIX mount point of the CD-ROM
- *%L* is the locale of the documentation that you want to use, for example, en_US

Updating the HTML documentation on your computer

When IBM makes updates available, you can update the HTML files that you installed from the DB2 HTML Documentation CD by either:

- Using the Information Center (if you have the DB2 administration graphical user interface tools installed)
- Downloading and applying a DB2 HTML Documentation fix pack

This procedure does not update the DB2 code.

Prerequisites:

Ensure that your computer has access to the Internet, because the updater downloads the latest documentation fix pack from the IBM server, if required. To connect to the Internet, you might need to supply your proxy information.

Procedure:

To use the information center to update your local HTML documentation:

1. Start the DB2 Information Center:

- From the graphical administration tools, click the **Information Center** icon in the toolbar.
 - At the command line, enter db2ic.
2. Click **Information Center** → **Update Local Documentation** to start the update.

If a documentation update is available, it is downloaded and applied.

To manually download and apply the documentation update:

1. Open the DB2 support page in your Web browser at www.ibm.com/software/data/db2/udb/winos2unix/support.
2. Click **DB2 Version 8** and find the documentation fix pack link for your operating system.
3. Determine if your local DB2 documentation is out of date by comparing the documentation fix pack level to the documentation level that you have installed.
4. If a more recent version of the documentation is available, then download the fix pack for your operating system. There is one fix pack for all Windows operating systems, and one fix pack for all UNIX operating systems.
5. Apply the fix pack:
 - For Windows operating systems: The documentation fix pack is a self-extracting zip file. Download the documentation fix pack to an empty directory and unzip it. Run the **setup** command in that directory to install the documentation fix pack.
 - For UNIX operating systems: The documentation fix pack is a compressed tar.Z file. Uncompress and untar the file to create a directory named `delta_install`. Run the script `installdocfix` inside that directory to install the documentation fix pack.

Searching the DB2 documentation

To search the DB2 documentation, use Netscape Version 6.1 (or later) or Microsoft Internet Explorer Version 5 (or later). Ensure that your browser's Java support is enabled.

A search window opens when you click the search icon in the navigation toolbar of the DB2 Information Integrator Information Center in a browser. If you are using the search function for the first time, it might take a minute or so for the search window to load.

Restrictions:

The following restrictions apply when you search the documentation:

- Boolean searches are not supported. The Boolean search qualifiers *and* and *or* are ignored in a search. For example, the following searches produce the same results:
 - servlets *and* beans
 - servlets *or* beans
- Wildcard searches are not supported. A search on *java** will look for the literal string *java** and will not, for example, find *javadoc*.

In general, you will get better search results if you search for phrases instead of single words.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest, then click **Search**. The **Results** field displays a list of topics that are ranked by accuracy.
Enter more terms to increase the precision of your query and reduce the number of topics that are returned by your query.
3. In the **Results** field, click the title of the topic that you want to read. The topic displays in the content frame.

When you perform a search, the first result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in results lists.

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by Web browsers. This topic provides possible solutions.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following solution, especially if you see the following line in the browser's Java console:

```
Cannot find class java/security/InvalidParameterException
```

Copy the following file from the DB2 HTML Documentation CD to the `java\classes\java\security\` directory within the directory where your Netscape browser is installed. You might need to create the `java\security\` subdirectory structure.

- On Windows operating systems:
`x:Program Files\IBM\sql1lib\doc\htmlcd\%L\InvalidParameterException.class`

- On UNIX operating systems:
 /*cdrom*/Program Files/IBM/sqllib/doc/htmlcd/%L
 /InvalidParameterException.class

where:

- *x* represents the Windows CD-ROM drive letter
- *cdrom* refers to the UNIX mount point of the CD-ROM
- %L is the locale of the documentation that you want to use, for example, en_US

If your Netscape browser still fails to display the search input window, try the following actions:

- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the computer. Then open a new instance of the Netscape browser and start the search again.
- Purge the browser's cache.
- Try a different version of Netscape, or a different browser.

Accessibility

Users with physical disabilities, such as restricted mobility or limited vision, can use software products successfully by using accessibility features. These are the major accessibility features in DB2 Information Integrator Version 8:

- You can operate all features by using the keyboard instead of the mouse.
- You can customize the size and color of your fonts.
- You can receive either visual or audio alert cues.
- DB2 supports accessibility applications that use the Java™ Accessibility API.
- DB2 documentation is provided in an accessible format.

Keyboard input and navigation

You can operate the DB2 database tools, such as Control Center, Data Warehouse Center, and Replication Center, by using only the keyboard. You can use keys or key combinations instead of a mouse to perform most operations.

In UNIX-based systems, the position of the keyboard focus is highlighted. This highlighting indicates which area of the window is active and where your keystrokes will have an effect.

Accessible display

The DB2 database tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font settings

For the DB2 database tools, you can use the Tools Settings notebook to select the color, size, and font for the text in menus and windows.

Nondependence on color

You do not need to distinguish between colors to use any of the functions in this product.

Alternative alert cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with assistive technologies

The DB2 Information Integrator graphical interface supports the Java Accessibility API, enabling the use of screen readers and other assistive technologies that are used by people with disabilities.

Accessible documentation

Documentation for the DB2 family of products is available in HTML format. You can view documentation according to the display preferences set in your browser. You can use screen readers and other assistive technologies.

Notices

This information was developed for products and services offered in the U.S.A. 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/region 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/region 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.

Licenses 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 that has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

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 document 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 illustrate 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 are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
DB2

The following terms are trademarks or registered trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

Index

C

C++ API

catalog classes

- Catalog_Option 2
- Column_Info 30
- list 1
- Nickname_Info 52
- Server_Info 13
- User_Info 24
- Wrapper_Info 4

data classes

- list 195
- Runtime_Data 203
- Runtime_Data_Desc 195
- Runtime_Data_Desc_List 200
- Runtime_Data_List 212

nickname classes

- Fenced_Generic_Nickname 119
- list 109
- Unfenced_Generic_Nickname 110

operation classes

- list 131
- Remote_Passthru 142
- Remote_Query 132

request classes

- list 149
- Predicate_List 188
- Reply 156
- Request 150
- Request_Constant 184
- Request_Exp 173
- Request_Exp_Type 180

server classes

- Fenced_Generic_Server 93
- list 81
- Unfenced_Generic_Server 82

user classes

- Fenced_Generic_User 106
- list 100
- Unfenced_Generic_User 100

wrapper classes

- Fenced_Generic_Wrapper 75
- list 67
- Unfenced_Generic_Wrapper 67

C++ classes and methods 1

catalog classes

- Catalog_Option 2
- Column_Info 30
- list 1
- Nickname_Info 52
- Server_Info 13
- User_Info 24
- Wrapper_Info 4

Catalog_Option class 2

Column_Info class 30

constructors

- Column_Info class 30
- Fenced_Generic_Nickname class 119
- Fenced_Generic_Wrapper class 75
- Fenced_Generic_Server class 93
- Fenced_Generic_User class 106
- Nickname_Info class 52
- Remote_Connection class 124
- Remote_Passthru class 142
- Remote_Query class 132
- Reply class 156
- Runtime_Data_Desc class 195
- Server_Info class 13
- Unfenced_Generic_Nickname class 110
- Unfenced_Generic_Wrapper class 67
- Unfenced_Generic_Server class 82
- Unfenced_Generic_User class 100
- User_Info class 24
- Wrapper_Info class 4

D

data classes

- list 195
- Runtime_Data 203
- Runtime_Data_Desc 195
- Runtime_Data_Desc_List 200
- Runtime_Data_List 212

data members

- Fenced_Generic_Nickname class 119
- Fenced_Generic_Server class 93
- Fenced_Generic_User class 106
- Remote_Connection class 124

data members (continued)

- Unfenced_Generic_Nickname class 110
- Unfenced_Generic_Wrapper class 67
- Unfenced_Generic_Server class 82
- Unfenced_Generic_User class 100

destructors

- Fenced_Generic_Wrapper class 75
- Fenced_Generic_User class 106
- Unfenced_Generic_Wrapper class 67
- Unfenced_Generic_User class 100

F

- Fenced_Generic_Nickname class 119
- Fenced_Generic_Server class 93
- Fenced_Generic_User class 106
- Fenced_Generic_Wrapper class 75

M

member functions

- Catalog_Option class 2
- Column_Info class 30
- Fenced_Generic_Nickname class 119
- Fenced_Generic_Wrapper class 75
- Fenced_Generic_Server class 93
- Fenced_Generic_User class 106
- Nickname_Info class 52
- Predicate_List class 188
- Remote_Connection class 124
- Remote_Passthru class 142
- Remote_Query class 132
- Reply class 156
- Request class 150
- Request_Constant class 184
- Request_Exp class 173
- Request_Exp_Type class 180
- Runtime_Data class 203
- Runtime_Data_Desc class 195
- Runtime_Data_Desc_List class 200
- Runtime_Data_List class 212

member functions (*continued*)
Server_Info class 13
Unfenced_Generic_Nickname
class 110
Unfenced_Generic_Wrapper
class 67
Unfenced_Generic_Server
class 82
Unfenced_Generic_User
class 100
User_Info class 24
Wrapper_Info class 4
Wrapper_Uilities class 214

N

nickname classes
Fenced_Generic_Nickname 119
list 109
Unfenced_Generic_
Nickname 110
Nickname_Info class 52

O

operation classes
list 131
Remote_Passthru 142
Remote_Query 132

P

Predicate_List class 188

R

Remote_Connection class 124
Remote_Passthru class 142
Remote_Query class 132
Reply class 156
Request class 150
request classes
list 149
Predicate_List 188
Reply 156
Request 150
Request_Constant 184
Request_Exp 173
Request_Exp_Type 180
Request_Constant class 184
Request_Exp class 173
Request_Exp_Type class 180
Runtime_Data class 203
Runtime_Data_Desc class 195
Runtime_Data_Desc_List class 200
Runtime_Data_List class 212

S

server classes
Fenced_Generic_Server 93

server classes (*continued*)
list 81
Unfenced_Generic_Server 82
Server_Info class 13

U

Unfenced_Generic_Nickname
class 110
Unfenced_Generic_Server class 82
Unfenced_Generic_User class 100
Unfenced_Generic_Wrapper
class 67
user classes
Fenced_Generic_User 106
list 100
Unfenced_Generic_User 100
User_Info class 24

W

wrapper classes
Fenced_Generic_Wrapper
class 75
list 67
Unfenced_Generic_Wrapper 67
Wrapper_Info class 4
Wrapper_Uilities class 214

Contacting IBM

To contact IBM in the United States or Canada, call one of the following numbers:

- For customer service: 1-800-IBM-SERV (1-800-426-7378)
- For DB2 marketing and sales: 1-800-IBM-4YOU (1-800-426-4968)

To learn about available service options, call one of the following numbers:

- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

To locate an IBM office in your country or region, see the IBM Directory of Worldwide Contacts on the Web at www.ibm.com/planetwide.

Product information

Information about DB2 Information Integrator is available by telephone or on the Web.

If you live in the United States, you can call one of the following numbers:

- To order products or to obtain general information: 1-800-IBM-CALL (1-800-426-2255)
- To order publications: 1-800-879-2755

On the Web, go to www.ibm.com/software/data/integration. This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, fix packs, news, and links to Web resources.

To locate an IBM office in your country or region, see the IBM Directory of Worldwide Contacts on the Web at www.ibm.com/planetwide.

Comments on the documentation

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Information Integrator documentation. You can use any of the following methods to provide comments:

- Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
- Send your comments by electronic mail (e-mail) to comments@us.ibm.com. Be sure to include the name of the product, the version number of the

product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).



Printed in U.S.A.

Spine information:



IBM DB2 Information
Integrator

Wrapper Developer's API Reference

Version 8