IBM DB2 Information Integrator

ラッパー開発者向け API リファレンス

バージョン 8

IBM DB2 Information Integrator

ラッパー開発者向け API リファレンス

バージョン 8

ご注意

本書および本書で紹介する製品をご使用になる前に、 229 ページの『特記事項』に記載されている情報をお読みください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： DB2 Information Integrator
Wrapper Developer's API Reference
Version 8

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2003.7

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について v
本書の対象読者 v
本書の表記規則および用語 v

C++ クラスおよびメソッド 1
C++ API のカタログ・クラス 1
Catalog_Option クラス 2
Wrapper_Info クラス. 4
Server_Info クラス 13
User_Info クラス 23
Column_Info クラス 30
Nickname_Info クラス. 49
C++ API のラッパー・クラス 64
Unfenced_Generic_Wrapper クラス 65
Fenced_Generic_Wrapper クラス 72
C++ API のサーバー・クラス 78
Unfenced_Generic_Server クラス 79
Fenced_Generic_Server クラス 90
C++ API のユーザー・クラス 97
Unfenced_Generic_User クラス 97
Fenced_Generic_User クラス 103
C++ API のニックネーム・クラス 106
Unfenced_Generic_Nickname クラス 107
Fenced_Generic_Nickname クラス. 115
Remote_Connection クラス 121
C++ API の操作クラス 128
Remote_Query クラス 128
Remote_Passthru クラス 138
C++ API の要求クラス 144
Request クラス. 145
Reply クラス 152
Request_Exp クラス 168
Request_Exp_Type クラス 174
Request_Constant クラス. 177
Predicate_List クラス 181
C++ API のデータ・クラス 187
Runtime_Data_Desc クラス 188
Runtime_Data_Desc_List クラス 192
Runtime_Data クラス. 195
Runtime_Data_List クラス 203
Wrapper_Utilities クラス. 205

DB2 Information Integrator の技術資料 215
資料およびリリース情報へのアクセス . . . 215
DB2 Information Integrator の資料 . . . 215
リリース情報およびインストール要件 . . 218
DB2 Information Integrator ドキュメンテーション・フィックスパック. 218
DB2 Information Integrator インフォメーション・センターまたは DB2 HTML ドキュメンテーション CD を使ったトピックへのアクセス 219
DB2 Information Integrator インフォメーション・センターの機能 219
DB2 Information Integrator インフォメーション・センターのトピックを見つける . . 219
DB2 HTML ドキュメンテーションの使用 221
DB2 文書の検索 224
Netscape 4.x を使って DB2 資料を検索する場合のトラブルシューティング . . . 225

アクセシビリティー. 227
キーボードによる入力およびナビゲーション 227
アクセスしやすい表示 227
フォントの設定 227
色覚への非依存 227
代替アラート・キュー 228
支援テクノロジーとの互換性 228
入手可能な資料 228

特記事項 229
商標 231

索引 233

IBM と連絡を取る 235
製品情報. 235

本書について

本書では、データ・ソースのラッパーを作成する際に使用可能な API クラスについて解説します。各クラスについて、概要と使用法を記載します。また、各クラスのメンバー関数 (および該当するコンストラクターとデストラクター) の目的、構文、戻り値、および必須入出力引き数をリストします。データ・ソースのラッパーを作成すると、そのデータ・ソースをフェデレーテッド (連合)・データベース・システムで使用できるようになります。

本書の対象読者

本書は、IBM® が提供する DB2® Information Integrator で API を使用する、DBA およびラッパー開発者を対象としています。

本書の表記規則および用語

強調表示規則:

本書では、以下の強調表示規則を使用します。

太文字

コマンドおよびグラフィカル・ユーザー・インターフェース・コントロール (フィールド名、プッシュボタン名、メニュー選択など) を示します。また、注意事項、制約事項、前提条件、および推奨事項を示すためにも使用します。

モノスペース

ユーザーが入力するテキスト、ファイル名、およびコード例を示します。また、SQL ステートメントや DB2 コマンド・パラメーター名にも使用します。

イタリック

ユーザーが適切な値で置き換える SQL ステートメントまたは DB2 コマンド・パラメーター値を示します。SQL ステートメントまたは DB2 コマンドの例では、パラメーター値の例にイタリックを使用します。また、用語の強調、新出用語の識別、および資料名の明示目的でも使用します。

大文字

DB2 コマンドと SQL ステートメントの名前、およびそれぞれのキーワードに使用します。また、データ・タイプ名、オプション、および頭字語にも使用します。

C++ クラスおよびメソッド

以下のセクションでは、C++ API で使用可能なクラスについて説明します。クラスには、以下のようなものがあります。

- カタログ・クラス
- ラッパー・クラス
- サーバー・クラス
- ユーザー・クラス
- ニックネーム・クラス
- リモート接続クラス
- 操作クラス
- 要求クラス
- データ・クラス
- ラッパー・ユーティリティー・クラス

各クラスについて、概要と使用法を記載します。また、各クラスのメンバー関数 (および該当するコンストラクターとデストラクター) の目的、構文、戻り値、および必須入出力引き数をリストします。

C++ API のカタログ・クラス

以下の表で、C++ API の各カタログ・クラスについて説明します。

表 1. カタログ・クラス

クラス名	説明
Catalog_Option	単一または複数の値で指定される特定のカタログ・オプションを表すクラス。サブクラスがこれらを識別します。このクラスは、インスタンス化またはサブクラス化しないでください。
Wrapper_Info	Catalog_Info のサブクラスであり、ラッパーのカタログ情報を持ちます。
Server_Info	サーバー・オブジェクトのカタログ情報 (CREATE SERVER ステートメントおよび ALTER SERVER ステートメントからのデータ) をカプセル化するクラス。

表 1. カタログ・クラス (続き)

クラス名	説明
User_Info	CREATE USER MAPPING ステートメントおよび ALTER USER MAPPING ステートメントからマップするユーザーのカタログ情報をカプセル化するクラス。
Column_Info	ニックネームの列のカタログ情報をカプセル化するクラス。このクラスは、列の統計情報を含みます。
Nickname_Info	カタログのニックネーム定義をカプセル化し、列および索引の定義を含むクラス。

関連情報:

- 2 ページの『Catalog_Option クラス』
- 4 ページの『Wrapper_Info クラス』
- 13 ページの『Server_Info クラス』
- 23 ページの『User_Info クラス』
- 30 ページの『Column_Info クラス』
- 49 ページの『Nickname_Info クラス』

Catalog_Option クラス

このセクションでは、Catalog_Option クラスについて説明し、その各メンバー関数について解説します。

概要

Catalog_Option クラスは、単一または複数の値で指定される特定のカタログ・オプションを表します。サブクラスがこれらを識別します。

Catalog_Option クラスは、C++ API のカタログ・クラスの 1 つです。

使用法 DB2 フェデレーテッド (連合)・サーバーが、カタログまたはデータ定義言語 (DDL) ステートメントで指定された各オプションについて、特定のカタログ・オプションをインスタンス化します。ラッパーはこれら特定のカタログ・オプションを Catalog_Info サブクラスの add_option() メソッドを使用してインスタンス化し、オプション値を追加または変更します。

ファイル

sqlqg_catalog.h

データ・メンバー

なし。

メンバー関数

以下の表で、Catalog_Option クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 2. Catalog_Option クラスのメンバー関数

メンバー関数	説明
get_name	オプション名を戻します。
get_action	このオプションのアクション (ADD、SET、 DROP、または、なし) を戻します。
get_value	オプション値を戻します。
get_value	オプション値および長さを戻します。

get_name 関数

目的 オプション名を戻します。

構文

```
sqluint8* get_name ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 ヌル終了オプション名。

get_action 関数

目的 このオプションのアクション (ADD、SET、DROP、または、なし) を戻します。

構文

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

入力引き数

なし。

出力引き数

なし。

戻り値 アクション。

get_value 関数

目的 オプション値を戻します。

構文

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

入力引き数

なし。

出力引き数

なし。

戻り値 ヌル終了オプション値。

get_value 関数

目的 オプション値および長さを戻します。

構文

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

入力引き数

なし。

出力引き数

表 3. get_value メンバー関数の出力引き数

名前	データ・タイプ	説明
a_length	sqlint32*	オプション値の長さ。

戻り値 ヌル終了オプション値。

関連情報:

- 1 ページの『C++ API のカタログ・クラス』

Wrapper_Info クラス

このセクションでは、Wrapper_Info クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Wrapper_Info クラスは、Catalog_Info のサブクラスであり、ラッパーのカタログ情報を持ちます。

Wrapper_Info クラスは、C++ API のカタログ・クラスの 1 つです。

使用法 このクラスは DB2 フェデレーテッド (連合)・サーバーによりインスタンス化され、CREATE WRAPPER ステートメントまたは DB2 Information Integrator カタログからの情報を持ちます。このクラスは CREATE WRAPPER ステートメントまたは ALTER WRAPPER ステートメントの処理時に情報が追加された場合に、ラッパーによりインスタンス化されます。

ファイル

sqlqg_catalog.h

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、Wrapper_Info クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 4. Wrapper_Info クラスのコンストラクター

コンストラクター	説明
Wrapper_Info	Wrapper_Info のインスタンスを構成します。

表 5. Wrapper_Info クラスのメンバー関数

メンバー関数	説明
get_wrapper_name	ラッパー名を戻します。
get_corelib	ラッパー・ライブラリーの名前を戻します。
set_type	ラッパー・タイプを設定します。
get_type	ラッパー・タイプを戻します。
set_version	ラッパー・コードのバージョンを設定します。
get_version	存在する場合は、ラッパー・バージョンを戻します。
copy	Wrapper_Info オブジェクトを複写します。
add_option	カタログ情報にオプションを追加します。
drop_option	Catalog_Info クラスからオプションを削除します。
get_option	カタログ・オプション名を戻します。

表 5. Wrapper_Info クラスのメンバー関数 (続き)

メンバー関数	説明
get_first_option	オプション・チェーンの最初のオプションにポインターを戻します。
get_next_option	オプション・チェーンの次のオプションを戻します。

Wrapper_Info コンストラクター

目的 Wrapper_Info のインスタンスを構成します。

構文

```
Wrapper_Info ()
```

入力引き数

なし。

出力引き数

なし。

get_wrapper_name 関数

目的 ラッパー名を戻します。

構文

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

入力引き数

なし。

出力引き数

表 6. get_wrapper_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_name	sqluint8**	ヌル終了ラッパー名ストリングへのポインター。

戻り値 戻りコード。名前が存在する場合は 0。名前が存在しない場合は SQLQG_NOVALUE。

get_corelib 関数

目的 ラッパー・ライブラリーの名前を戻します。この名前は、CREATE WRAPPER ステートメントからのラッパー・ライブラリーのベース名です。

構文

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

入力引き数

なし。

出力引き数

表 7. get_corelib メンバー関数の出力引き数

名前	データ・タイプ	説明
a_lib_name	sqluint8**	ヌル終了ライブラリー名ストリングへのポインター。

戻り値 戻りコード。名前が存在する場合は 0。名前が存在しない場合は SQLQG_NOVALUE。

set_type 関数

目的 ラッパー・タイプを設定します。

構文

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

入力引き数

表 8. set_type メンバー関数の入力引き数

名前	データ・タイプ	説明
a_wrapper_type	sqluint8	ラッパー・タイプ (常に N)。

出力引き数

なし。

戻り値 なし。

get_type 関数

目的 ラッパー・タイプを戻します。

構文

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

入力引き数

なし。

出力引き数

表 9. get_type メンバー関数の出力引き数

名前	データ・タイプ	説明
a_wrapper_type	sqluint8*	ラッパー・タイプ。

戻り値 戻りコード。タイプが存在する場合は 0。タイプが存在しない場合は SQLQG_NOVALUE。

set_version 関数

目的 ラッパー・コードのバージョンを設定します。

構文

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

入力引き数

表 10. set_version メンバー関数の入力引き数

名前	データ・タイプ	説明
a_wrapper_version	sqlint32	ラッパー・コードのバージョン。

出力引き数

なし。

戻り値 なし。

get_version 関数

目的 存在する場合は、ラッパー・バージョンを戻します。

構文

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

入力引き数

なし。

出力引き数

表 11. get_version メンバー関数の出力引き数

名前	データ・タイプ	説明
a_wrapper_version	sqlint32*	ラッパー・コードのバージョン。

戻り値 戻りコード。バージョンが存在する場合は 0。バージョンが存在しない場合は SQLQG_NOVALUE。

copy 関数

目的 Wrapper_Info オブジェクトを複写します。

構文

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

入力引き数

なし。

出力引き数

表 12. copy メンバー関数の出力引き数

名前	データ・タイプ	説明
a_new_wrapper_info	Wrapper_Info**	新しく割り振った Wrapper_Info へのポインター。

戻り値 戻りコード。0 は成功を示します。

add_option 関数

定義場所

Catalog_Info

目的 カタログ情報にオプションを追加します。

使用法 ラッパーは、CREATE WRAPPER ステートメントおよび ALTER WRAPPER ステートメントの処理中にラッパー生成オプションがカタログに追加された場合、このメンバー関数を呼び出すことができます。

構文

```
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 = "")
```

入力引き数

表 13. add_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_name_len	sqlint32	オプション名の長さ。
a_opt_value	sqluint8*	オプション値 (非ヌル終了)。
a_value_len	sqlint32	オプション値の長さ。
a_action	Catalog_Option::Action	このオプションのアクション (ADD、SET、DROP、または、なし)。
a_option_type	char*	重複オプションである場合は、SQLN1884 エラー・メッセージに使用されるトークン。ラッパー・オプションについては、sqlqg_misc.h ヘッダー・ファイルで定義された SQLQG_WRAPPER_OPTION 定数を使用します。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

drop_option 関数

定義場所

Catalog_Info

目的 Catalog_Info クラスからオプションを削除します。このメンバー関数は、カタログからオプションを削除しません。オプションは、デルタ Catalog_Info オブジェクトに対してオプションが追加された場合に、Catalog_Option::sqlqg_Drop のアクションにより削除されます。

構文

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

入力引き数

表 14. drop_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option*	削除するオプション。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_option 関数

定義場所

Catalog_Info

目的 カタログ・オプション名を戻します。

使用法 オプションが検出されない場合は、出力引き数 `option` が NULL になります。

構文

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

入力引き数

表 15. get_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_name_len	sqlint32	オプション名の長さ。

出力引き数

表 16. get_option メンバー関数の出力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option**	検出されたオプション。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は、オプションが検出されなかったことを示します。

get_first_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの最初のオプションにポインターを戻します。

構文

```
Catalog_Option* get_first_option ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 オプション・チェーンの最初のオプションへのポインター。チェーンが空の場合は NULL。

get_next_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの次のオプションを戻します。

構文

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

入力引き数

表 17. get_next_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_current_option	Catalog_Option*	現行オプション。

出力引き数

なし。

戻り値 オプション・チェーンの次のオプションへのポインター。チェーンの末尾の場合は NULL。

関連情報:

- 1 ページの『C++ API のカタログ・クラス』

Server_Info クラス

このセクションでは、Server_Info クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Server_Info クラスは、サーバー・オブジェクトのカタログ情報 (CREATE SERVER ステートメントおよび ALTER SERVER ステートメントからのデータ) をカプセル化します。

Server_Info クラスは、C++ API のカタログ・クラスの 1 つです。

使用法 このクラスは DB2 フェデレーテッド (連合)・サーバーによりインスタンス化され、CREATE SERVER ステートメントまたは ALTER SERVER ステートメント、あるいは DB2 Information Integrator カタログからの情報を持ちます。また、このクラスは CREATE SERVER ステートメントまたは ALTER SERVER ステートメントの処理時に情報が追加された場合に、ラッパーによりインスタンス化されます。

ファイル

sqlqg_catalog.h

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、Server_Info クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 18. Server_Info クラスのコンストラクター

コンストラクター	説明
Server_Info	空の Server_Info オブジェクトを構成します。
Server_Info	指定されたパラメーターを使用して Server_Info オブジェクトを構成します。

表 19. Server_Info クラスのメンバー関数

メンバー関数	説明
add_option	カタログ情報に単一値オプションを追加します。
drop_option	Catalog_Info クラスからオプションを削除します。

表 19. Server_Info クラスのメンバー関数 (続き)

メンバー関数	説明
get_option	カタログ・オプション名を戻します。
get_first_option	オプション・チェーンの最初のオプションにポインターを戻します。
get_next_option	オプション・チェーンの次のオプションを戻します。
get_basic_info	Server_Info オブジェクトからの基本情報を戻します。
get_server_name	Server_Info からサーバー名 (有効な場合) を戻します。
set_server_type	サーバーのサーバー・タイプを設定します。
get_server_type	Server_Info からサーバー・タイプ (有効な場合) を戻します。
set_server_version	サーバーのサーバー・バージョンを設定します。
get_server_version	Server_Info からサーバー・バージョン (有効な場合) を戻します。
get_wrapper_name	Server_Info からラッパー名 (有効な場合) を戻します。
merge	デルタ Server_Info オブジェクトを現行オブジェクトにマージします。
copy	Server_Info オブジェクトを複写します。

Server_Info コンストラクター

目的 空の Server_Info オブジェクトを構成します。

構文

```
Server_Info ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 なし。

Server_Info コンストラクター

目的 指定されたパラメーターを使用して Server_Info オブジェクトを構成します。

構文

```
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);
```

入力引き数

表 20. Server_Info コンストラクターの入力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8*	サーバー名 (非ヌル終了)。
a_name_len	sqlint32	サーバー名の長さ。
a_server_type	sqluint8*	サーバー・タイプ (非ヌル終了)。
a_type_len	sqlint32	サーバー・タイプの長さ。
a_server_version	sqluint8*	サーバー・バージョン (非ヌル終了)。
a_ver_len	sqlint32	サーバー・バージョンの長さ。
a_server_wrapper	sqluint8*	ラッパー名 (非ヌル終了)。
a_wrap_len	sqlint32	ラッパー名の長さ。

出力引き数

表 21. Server_info コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 なし。

add_option 関数

定義場所

Catalog_Info

目的 カタログ情報に単一値オプションを追加します。

使用法 ラッパーは、CREATE SERVER ステートメントまたは ALTER SERVER ステートメントの処理中にラッパー生成オプションがカタログに追加された場合、このメンバー関数を呼び出すことができます。

構文

```
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 = "")
```

入力引き数

表 22. add_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_opt_name_len	sqlint32	オプション名の長さ。
a_opt_value	sqluint8*	オプション値 (非ヌル終了)。
a_opt_value_len	sqlint32	オプション値の長さ。
a_act	Catalog_Option::Action	このオプションのアクション (ADD、SET、DROP、または、なし)。
a_opt_type	char*	重複オプションである場合は、SQLN1884 エラー・メッセージに使用されるトークン。sqlqg_misc.h ヘッダー・ファイルで定義された SQLQG_SERVER_OPTION 定数を使用します。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

drop_option 関数

定義場所

Catalog_Info

目的 Catalog_Info クラスからオプションを削除します。このメンバー関数は、カタログからオプションを削除しません。オプションは、デルタ Catalog_Info オブジェクトに対してオプションが追加された場合に、Catalog_Option::sqlqg_Drop のアクションにより削除されます。

構文

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

入力引き数

表 23. drop_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option*	削除するオプション。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_option 関数

定義場所

Catalog_Info

目的 カタログ・オプション名を戻します。

使用法 オプションが検出されない場合は、出力引き数 `option` が NULL になります。

構文

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

入力引き数

表 24. get_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_name_len	sqlint32	オプション名の長さ。

出力引き数

表 25. get_option メンバー関数の出力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option**	検出されたオプション。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は、オプションが検出されなかったことを示します。

get_first_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの最初のオプションにポインターを戻します。

構文

```
Catalog_Option* get_first_option ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 オプション・チェーンの最初のオプションへのポインター。チェーンが空の場合は NULL。

get_next_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの次のオプションを戻します。

構文

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

入力引き数

表 26. get_next_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_current_option	Catalog_Option*	現行オプション。

出力引き数
なし。

戻り値 チェーンの次のオプションへのポインター。チェーンの末尾の場合は NULL。

get_basic_info 関数

目的 Server_Info オブジェクトからの基本情報を戻します。他の get_XXX メンバー・ルーチンとは異なり、get_basic_info メンバー関数は、無効なデータ項目がある場合 SQLQG_ERROR を戻し、エラーを記録します。

構文

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

入力引き数
なし。

出力引き数

表 27. get_basic_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8**	ヌル終了サーバー名へのポインター。
a_server_type	sqluint8**	ヌル終了サーバー・タイプへのポインター。
a_server_version	sqluint8**	ヌル終了サーバー・バージョンへのポインター。
a_server_wrapper	sqluint8**	ヌル終了ラッパー名へのポインター。

戻り値 戻りコード。0 は成功を示します。

get_server_name 関数

目的 Server_Info からサーバー名 (有効な場合) を戻します。

構文

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

入力引き数
なし。

出力引き数

表 28. get_server_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8**	ヌル終了サーバー名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE はサーバー名が設定されていないことを示します。

set_server_type 関数

目的 サーバーのサーバー・タイプを設定します。

使用法 CREATE SERVER ステートメントまたは ALTER SERVER ステートメントを処理する際にデフォルト値が指定されていない場合、ラッパーはこのメンバー関数を呼び出してデフォルト値を指定できます。

構文

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

入力引き数

表 29. set_server_type メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_type	sqluint8*	サーバー・タイプ (非ヌル終了)。
a_server_type_len	sqlint32	サーバー・タイプの長さ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_server_type 関数

目的 Server_Info からサーバー・タイプ (有効な場合) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 30. get_server_type メンバー関数の出力引き数

名前	データ・タイプ	説明
a_server_type	sqluint8**	ヌル終了サーバー・タイプへのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE はサーバー・タイプが設定されていないことを示します。

set_server_version 関数

目的 サーバーのサーバー・バージョンを設定します。

使用法 CREATE SERVER ステートメントまたは ALTER SERVER ステートメントを処理する際、DDL にデフォルトのサーバー・バージョンが指定されていない場合、ラッパーはこのメンバー関数を呼び出してデフォルトのサーバー・バージョンを指定できます。

構文

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

入力引き数

表 31. set_server_version メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_version	sqluint8*	サーバー・バージョン (非ヌル終了)。
a_server_version_len	sqlint32	サーバー・バージョンの長さ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_server_version 関数

目的 Server_Info からサーバー・バージョン (有効な場合) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 32. get_server_version メンバー関数の出力引き数

名前	データ・タイプ	説明
a_server_version	sqluint8**	ヌル終了サーバー・バージョンへのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE はサーバー・バージョンが設定されていないことを示します。

get_wrapper_name 関数

目的 Server_Info からラッパー名 (有効な場合) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 33. get_wrapper_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_wrapper_name	sqluint8**	ヌル終了ラッパー名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE はラッパー名が設定されていないことを示します。

merge 関数

目的 デルタ Server_Info オブジェクトを現行オブジェクトにマージします。

構文

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

入力引き数

表 34. merge メンバー関数の入力引き数

名前	データ・タイプ	説明
a_delta_info	Server_Info*	デルタ・オブジェクト。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

copy 関数

目的 Server_Info オブジェクトを複写します。

構文

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

入力引き数

なし。

出力引き数

表 35. copy メンバー関数の出力引き数

名前	データ・タイプ	説明
a_new_server_info	Server_Info**	新しい Server_Info オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 1 ページの『C++ API のカタログ・クラス』

User_Info クラス

このセクションでは、User_Info クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

User_Info クラスは、CREATE USER MAPPING ステートメントおよび ALTER USER MAPPING ステートメントからマップするユーザーのカタログ情報をカプセル化します。

User_Info クラスは、C++ API のカタログ・クラスの 1 つです。

使用法 このクラスは DB2 フェデレーテッド (連合)・サーバーによりインスタンス化され、CREATE USER MAPPING ステートメントまたは ALTER USER MAPPING ステートメント、あるいは DB2 Information Integrator カタログからの情報を持ちます。このクラスは、CREATE USER MAPPING ステートメントまたは ALTER MAPPING ステートメントの処理時に情報が追加された場合に、ラッパーによりインスタンス化されます。

ファイル

sqlqg_catalog.h

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、User_Info クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 36. User_Info クラスのコンストラクター

コンストラクター	説明
User_Info	デフォルトの (空) User_Info オブジェクトを構成します。

表 37. User_Info クラスのメンバー関数

メンバー関数	説明
get_server_name	このユーザー・マッピングに対するサーバー名を戻します。
get_authid	このユーザー・マッピングに対する許可 ID を戻します。
merge	デルタ User_Info オブジェクトを現行オブジェクトにマージします。
copy	現行 User_Info オブジェクトのコピーを作成します。
add_option	カタログ情報に単一値オプションを追加します。
drop_option	Catalog_Info クラスからオプションを削除します。
get_option	カタログ・オプション名を戻します。
get_first_option	オプション・チェーンの最初のオプションにポインターを戻します。

表 37. User_Info クラスのメンバー関数 (続き)

メンバー関数	説明
get_next_option	オプション・チェーンの次のオプションを戻します。

User_Info コンストラクター

目的 デフォルトの (空) User_Info オブジェクトを構成します。

構文

```
User_Info ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 なし。

get_server_name 関数

目的 このユーザー・マッピングに対するサーバー名を戻します。

構文

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

入力引き数

なし。

出力引き数

表 38. get_server_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8**	ヌル終了サーバー名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE はサーバー名が無効、または設定されていないことを示します。

get_authid 関数

目的 このユーザー・マッピングに対する許可 ID を戻します。

構文

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

入力引き数

なし。

出力引き数

表 39. get_authid メンバー関数の出力引き数

名前	データ・タイプ	説明
a_authid	sqluint8**	ヌル終了許可 ID へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は ID が無効、または設定されていないことを示します。

merge 関数

目的 デルタ User_Info オブジェクトを現行オブジェクトにマージします。

構文

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

入力引き数

表 40. merge メンバー関数の入力引き数

名前	データ・タイプ	説明
a_delta_info	User_Info*	マージするデルタ・オブジェクト。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

copy 関数

目的 現行 User_Info オブジェクトのコピーを作成します。

構文

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

入力引き数

なし。

出力引き数

表 41. copy メンバー関数の出力引き数

名前	データ・タイプ	説明
a_new_user_info	User_Info**	重複オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

add_option 関数

目的 カタログ情報に単一値オプションを追加します。

使用法 ラッパーはこのメンバー関数を CREATE USER MAPPING ステートメントまたは ALTER USER MAPPING ステートメントの処理中に呼び出し、ラッパー生成オプションを追加できます。

構文

```
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 = "")
```

入力引き数

表 42. add_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_opt_name_len	sqlint32	オプション名の長さ。
a_opt_value	sqluint8*	オプション値 (非ヌル終了)。
a_opt_value_len	sqlint32	オプション値の長さ。
a_act	Catalog_Option::Action	このオプションのアクション (ADD、SET、DROP、または、なし)。
a_opt_type	char*	重複オプションである場合は、SQLN1884 エラー・メッセージに使用されるトークン。sqlqg_misc.h ヘッダー・ファイルで定義された SQLQG_USER_OPTION 定数を使用します。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

drop_option 関数

定義場所

Catalog_Info

目的 Catalog_Info クラスからオプションを削除します。このメンバー関数は、カタログからオプションを削除しません。オプションは、デルタ Catalog_Info オブジェクトに対してオプションが追加された場合に、Catalog_Option::sqlqg_Drop のアクションにより削除されます。

構文

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

入力引き数

表 43. drop_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option*	削除するオプション。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_option 関数

定義場所

Catalog_Info

目的 カタログ・オプション名を戻します。

使用法 オプションが検出されない場合は、出力引き数 `option` が NULL になります。

構文

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

入力引き数

表 44. get_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_name_len	sqlint32	オプション名の長さ。

出力引き数

表 45. get_option メンバー関数の出力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option**	検出されたオプション。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は、オプションが検出されなかったことを示します。

get_first_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの最初のオプションにポインターを戻します。

構文

```
Catalog_Option* get_first_option ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 オプション・チェーンの最初のオプションへのポインター。チェーンが空の場合は NULL。

get_next_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの次のオプションを戻します。

構文

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

入力引き数

表 46. get_next_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_current_option	Catalog_Option*	現行オプション。

出力引き数
なし。

戻り値 チェーンの次のオプションへのポインター。チェーンの末尾の場合は NULL。

関連情報:

- 1 ページの『C++ API のカタログ・クラス』

Column_Info クラス

このセクションでは、Column_Info クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Column_Info クラスは、ニックネームの列のカタログ情報をカプセル化します。このクラスは、列の統計情報を含みます。

Column_Info クラスは、C++ API のカタログ・クラスの 1 つです。

使用法 このクラスは DB2 フェデレーテッド (連合)・サーバーによりインスタンス化され、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメント、あるいは DB2 Information Integrator カタログからの情報を持ちます。このクラスは CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理時に情報が追加された場合に、ラッパーによりインスタンス化されます。

ファイル
sqlqg_catalog.h

データ・メンバー
なし。

コンストラクターおよびメンバー関数

以下の表で、Column_Info クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 47. Column_Info クラスのコンストラクター

コンストラクター	説明
Column_Info	デフォルトの (空) Column_Info オブジェクトを構成します。

表 48. Column_Info クラスのメンバー関数

メンバー関数	説明
set_column_name	列名を設定します。
get_column_name	この列の名前を戻します。
get_new_column_name	列の名前を変更する ALTER (または SET) COLUMN 節を含む ALTER COLUMN ステートメントに指定された新しい列名を戻します。
set_column_type	列のタイプを設定します。
get_column_type	Server_Info からローカル列タイプ (有効な場合) を戻します。
set_for_bit_data	列に FOR BIT DATA フラグを設定します。
get_for_bit_data	列の FOR BIT DATA フラグ (有効な場合) を戻します。
set_column_ID	列の ID (位置) を設定します。
get_column_ID	列の ID (位置) (有効な場合) を戻します。
set_org_length	列の最大長を設定します。
get_org_length	値が有効な場合、Column_Info から列の最大長を戻します。
set_org_scale	列の数値尺度を設定します。
get_org_scale	値が有効な場合、Column_Info から数値尺度を戻します。
set_nulls	列に NULL 許可フラグを設定します。
get_nulls	値が有効な場合、Column_Info から NULL 許可フラグを戻します。
set_avg_length	列の平均長を設定します。
get_avg_length	値が有効な場合、Column_Info から列の平均長を戻します。
set_high2key	列の 2 番目に大きい値を設定します。
get_high2key	値が有効な場合、Column_Info から 2 番目に大きい値を戻します。

表 48. *Column_Info* クラスのメンバー関数 (続き)

メンバー関数	説明
set_low2key	列の 2 番目に小さい値を設定します。
get_low2key	値が有効な場合、Column_Info から 2 番目に小さい値を戻します。
get_default	値が有効な場合、Column_Info からデフォルト値を戻します。
set_colcard	列のカーディナリティーを設定します。
get_colcard	値が有効な場合、Column_Info からカーディナリティーを戻します。
set_codepage1	列のコード・ページを設定します。
get_codepage1	値が有効な場合、Column_Info からコード・ページを戻します。
set_codepage2	列のコード・ページを設定します。
get_codepage2	値が有効な場合、Column_Info からコード・ページを戻します。
merge	デルタ Column_Info オブジェクトを現行オブジェクトにマージします。
copy	Column_Info オブジェクトを複写します。
add_option	カタログ情報に単一値オプションを追加します。
drop_option	Catalog_Info クラスからオプションを削除します。
get_option	カタログ・オプション名を戻します。
get_first_option	オプション・チェーンの最初のオプションにポインターを戻します。
get_next_option	オプション・チェーンの次のオプションを戻します。

Column_Info コンストラクター

目的 デフォルトの (空) Column_Info オブジェクトを構成します。

構文

```
Column_Info ()
```

入力引き数

なし。

出力引き数
なし。

戻り値 なし。

set_column_name 関数

目的 列名を設定します。

使用法 列の名前変更にこのメンバー関数を使用しないでください。

構文

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

入力引き数

表 49. set_column_name メンバー関数の入力引き数

名前	データ・タイプ	説明
a_column_name	sqluint8*	列名 (非ヌル終了)。
a_column_name_len	sqlint32	列名の長さ。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

get_column_name 関数

目的 この列の名前を戻します。

構文

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

入力引き数
なし。

出力引き数

表 50. get_column_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_column_name	sqluint8**	ヌル終了列名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は列名が無効、または設定されていないことを示します。

get_new_column_name 関数

目的 列の名前を変更する ALTER (または SET) COLUMN 節を含む ALTER COLUMN ステートメントに指定された新しい列名を戻します。

構文

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

入力引き数

なし。

出力引き数

表 51. get_new_column_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_new_col_name	sqluint8**	新しいヌル終了列名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は新しい列名が設定されていないことを示します。

set_column_type 関数

目的 列のタイプを設定します。

構文

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

入力引き数

表 52. set_column_type メンバー関数の入力引き数

名前	データ・タイプ	説明
a_column_type	sqluint8*	列タイプ (非ヌル終了)。
a_column_type_len	sqlint32	列タイプの長さ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_column_type 関数

目的 Server_Info からローカル列タイプ (有効な場合) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 53. get_column_type メンバー関数の出力引き数

名前	データ・タイプ	説明
a_column_type	sqluint8**	ヌル終了列タイプへのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE はタイプが設定されていないことを示します。

set_for_bit_data 関数

目的 列に FOR BIT DATA フラグを設定します。

構文

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

入力引き数

表 54. set_for_bit_data メンバー関数の入力引き数

名前	データ・タイプ	説明
a_for_bit_data	sqluint8	FOR BIT DATA フラグ (Y または N)。

出力引き数

なし。

戻り値 なし。

get_for_bit_data 関数

目的 列の FOR BIT DATA フラグ (有効な場合) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 55. get_for_bit_data メンバー関数の出力引き数

名前	データ・タイプ	説明
a_for_bit_data	sqluint8*	フラグ Y または N。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE はフラグが設定されていないことを示します。

set_column_ID 関数

目的 列の ID (位置) を設定します。

使用法 ニックネーム内の列順序を変更するのに、このメンバー関数を使用しないでください。

構文

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

入力引き数

表 56. set_column_ID メンバー関数の入力引き数

名前	データ・タイプ	説明
a_column_ID	sqlint16	列 ID (位置)。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_column_ID 関数

目的 列の ID (位置) (有効な場合) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 57. get_column_ID メンバー関数の出力引き数

名前	データ・タイプ	説明
a_column_ID	sqluint16*	列 ID。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は ID が設定されていないことを示します。

set_org_length 関数

目的 列の最大長 (バイト) を設定します。

構文

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

入力引き数

表 58. set_org_length メンバー関数の入力引き数

名前	データ・タイプ	説明
a_org_length	sqlint32	列の最大長。

出力引き数

なし。

戻り値 なし。

get_org_length 関数

目的 値が有効な場合、Column_Info から列の最大長 (バイト) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 59. get_org_length メンバー関数の出力引き数

名前	データ・タイプ	説明
a_org_length	sqlint32*	列の最大長。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_org_scale 関数

目的 列の数値尺度を設定します。

構文

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

入力引き数

表 60. set_org_scale メンバー関数の入力引き数

名前	データ・タイプ	説明
a_org_scale	sqlint16	尺度。

出力引き数

なし。

戻り値 なし。

get_org_scale 関数

目的 値が有効な場合、Column_Info から数値尺度を戻します。

構文

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

入力引き数

なし。

出力引き数

表 61. get_org_scale メンバー関数の出力引き数

名前	データ・タイプ	説明
a_org_scale	sqlint16*	数値尺度。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_nulls 関数

目的 列に NULL 許可フラグを設定します。

構文

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

入力引き数

表 62. set_nulls メンバー関数の入力引き数

名前	データ・タイプ	説明
a_nulls	sqluint8	NULL 許可フラグ。フラグは Y または N。

出力引き数

なし。

戻り値 なし。

get_nulls 関数

目的 値が有効な場合、Column_Info から NULL 許可フラグを戻します。

構文

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

入力引き数

なし。

出力引き数

表 63. get_nulls メンバー関数の出力引き数

名前	データ・タイプ	説明
a_nulls	sqluint8*	列の NULL 許可フラグ。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_avg_length 関数

目的 列の平均長 (バイト) を設定します。

使用法 ラッパーは、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理中に、列の平均長を設定します。DB2 オプティマイザーは照会最適化プランを作成する際、この平均長の情報を使用します。

構文

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

入力引き数

表 64. set_avg_length メンバー関数の入力引き数

名前	データ・タイプ	説明
a_avg_len	sqlint32	列の平均長。

出力引き数

なし。

戻り値 なし。

get_avg_length 関数

目的 値が有効な場合、Column_Info から列の平均長 (バイト) を戻します。

構文

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

入力引き数

なし。

出力引き数

表 65. get_avg_length メンバー関数の出力引き数

名前	データ・タイプ	説明
a_avg_len	sqlint32*	列の平均長。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_high2key 関数

目的 列の 2 番目に大きい値を設定します。

使用法 ラッパーは、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理中に、列の 2 番目に大きい値を設定できます。DB2 オプティマイザーは、照会最適化プランの作成時に、この 2 番目に大きい値または最大値を使用できます。

構文

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

入力引き数

表 66. set_high2key メンバー関数の入力引き数

名前	データ・タイプ	説明
a_high2key	sqluint8*	2 番目に大きい文字ストリング値 (非ヌル終了)。
a_high2key_len	sqlint32	値の長さ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_high2key 関数

目的 値が有効な場合、Column_Info から 2 番目に大きい値を戻します。

構文

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

入力引き数

なし。

出力引き数

表 67. get_high2key メンバー関数の出力引き数

名前	データ・タイプ	説明
a_high2key	sqluint8**	列の 2 番目に大きい値。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_low2key 関数

目的 列の 2 番目に小さい値を設定します。

使用法 ラッパーは、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理中に、列の 2 番目に小さい値を設定できます。DB2 オプティマイザーは、照会最適化プランの作成時に、この 2 番目に小さい値または最小値を使用できます。

構文

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

入力引き数

表 68. set_low2key メンバー関数の入力引き数

名前	データ・タイプ	説明
a_low2key	sqluint8*	2 番目に小さい文字ストリング値 (非ヌル終了)。
a_low2key_len	sqlint32	値の長さ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_low2key 関数

目的 値が有効な場合、Column_Info から 2 番目に小さい値を戻します。

構文

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

入力引き数

なし。

出力引き数

表 69. get_low2key メンバー関数の出力引き数

名前	データ・タイプ	説明
a_low2key	sqluint8**	列の 2 番目に小さい値。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

get_default 関数

目的 値が有効な場合、Column_Info からデフォルト値を戻します。

構文

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

入力引き数

なし。

出力引き数

表 70. get_default メンバー関数の出力引き数

名前	データ・タイプ	説明
a_default	sqluint8**	列のデフォルト値。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_colcard 関数

目的 列のカーディナリティーを設定します。

使用法 ラッパーは、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理中に、列のカーディナリティー (既知の場合) を設定します。DB2 オプティマイザーは、最適パフォーマンス・プランの作成時にこの情報を使用します。固有の値を持つ列 (重複なし) については、列のカーディナリティーとニックネームのカーディナリティーが同じである必要があります。列のカーディナリティーがニックネームのカーディナリティーより大きい場合は、DB2 オプティマイザーがエラーを生成します。

構文

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

入力引き数

表 71. set_colcard メンバー関数の入力引き数

名前	データ・タイプ	説明
a_colcard	sqlint64	列のカーディナリティー。

出力引き数

なし。

戻り値 なし。

get_colcard 関数

目的 値が有効な場合、Column_Info からカーディナリティーを戻します。

構文

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

入力引き数

なし。

出力引き数

表 72. get_colcard メンバー関数の出力引き数

名前	データ・タイプ	説明
a_colcard	sqlint64*	列のカーディナリティー。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_codepage1 関数

目的 列のコード・ページを設定します。

構文

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

入力引き数

表 73. set_codepage1 メンバー関数の入力引き数

名前	データ・タイプ	説明
a_codepage1	sqlint16	列のコード・ページ。

出力引き数

なし。

戻り値 なし。

get_codepage1 関数

目的 値が有効な場合、Column_Info からコード・ページを戻します。

構文

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

入力引き数

なし。

出力引き数

表 74. get_codepage1 メンバー関数の出力引き数

名前	データ・タイプ	説明
a_codepage1	sqlint16*	列のコード・ページ。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_codepage2 関数

目的 列のコード・ページを設定します。

構文

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

入力引き数

表 75. set_codepage2 メンバー関数の入力引き数

名前	データ・タイプ	説明
a_codepage2	sqlint16	列のコード・ページ。

出力引き数

なし。

戻り値 なし。

get_codepage2 関数

目的 値が有効な場合、Column_Info からコード・ページを戻します。

構文

```
sqlint32 get_codepage2 (sqlint16* a_codepage2)
```

入力引き数

なし。

出力引き数

表 76. get_codepage2 メンバー関数の出力引き数

名前	データ・タイプ	説明
a_codepage2	sqlint16*	列のコード・ページ。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

merge 関数

目的 デルタ Column_Info オブジェクトを現行オブジェクトにマージします。

構文

```
sqlint32 merge (Column_Info* a_delta_info)
```

入力引き数

表 77. merge メンバー関数の入力引き数

名前	データ・タイプ	説明
a_delta_info	Column_Info*	マージするデータ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

copy 関数

目的 Column_Info オブジェクトを複写します。

構文

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

入力引き数

なし。

出力引き数

表 78. copy メンバー関数の出力引き数

名前	データ・タイプ	説明
a_new_column_info	Column_Info**	重複オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

add_option 関数

定義場所

Catalog_Info

目的 カタログ情報に単一値オプションを追加します。

使用法 ラッパーは、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理中にラッパー生成オプションがカタログに追加された場合、このメンバー関数を呼び出します。

構文

```
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 = "")
```

入力引き数

表 79. add_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_opt_name_len	sqlint32	オプション名の長さ。
a_opt_value	sqluint8*	オプション値 (非ヌル終了)。
a_opt_value_len	sqlint32	オプション値の長さ。
a_act	Catalog_Option::Action	このオプションのアクション (ADD、SET、DROP、または、なし)。
a_opt_type	char*	重複オプションである場合は、SQLN1884 エラー・メッセージに使用されるトークン。sqlqg_misc.h ヘッダー・ファイルで定義された SQLQG_COLUMN_OPTION 定数を使用します。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

drop_option 関数

定義場所

Catalog_Info

目的 Column_Info クラスからオプションを削除します。このメンバー関数は、カタログからオプションを削除しません。オプションは、デルタ Catalog_Info オブジェクトに対してオプションが追加された場合に、Catalog_Option::sqlqg_Drop のアクションにより削除されます。

構文

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

入力引き数

表 80. drop_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option*	削除するオプション。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_option 関数

定義場所

Catalog_Info

目的 カタログ・オプション名を戻します。

使用法 オプションが検出されない場合は、出力引き数 `a_option` が NULL になります。

構文

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

入力引き数

表 81. get_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_name_len	sqlint32	オプション名の長さ。

出力引き数

表 82. get_option メンバー関数の出力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option**	検出されたオプション。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は、オプションが検出されなかったことを示します。

get_first_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの最初のオプションにポインターを戻します。

構文

```
Catalog_Option* get_first_option ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 チェーンの最初のオプションへのポインター。チェーンが空の場合は NULL。

get_next_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの次のオプションを戻します。

構文

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

入力引き数

表 83. get_next_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_current_option	Catalog_Option*	現行オプション。

出力引き数

なし。

戻り値 チェーンの次のオプションへのポインター。チェーンの末尾の場合は NULL。

関連情報:

- 1 ページの『C++ API のカタログ・クラス』

Nickname_Info クラス

このセクションでは、Nickname_Info クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Nickname_Info クラスは、カタログのニックネーム定義をカプセル化し、列の定義を含みます。

Nickname_Info クラスは、C++ API のカタログ・クラスの 1 つです。

使用法 このクラスは DB2 フェデレーテッド (連合)・サーバーによりインスタンス化され、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメント、あるいは DB2 Information Integrator カタログからの情報を持ちます。このクラスは CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理時に情報が追加された場合に、ラッパーによりインスタンス化されます。

ファイル

sqlqg_catalog.h

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、Nickname_Info クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 84. Nickname_Info クラスのコンストラクター

コンストラクター	説明
Nickname_Info	デフォルトの (空) Nickname_Info オブジェクトを構成します。

表 85. Nickname_Info クラスのメンバー関数

メンバー関数	説明
get_local_schema	ニックネームのローカル・スキーマ名を戻します。
get_nickname	ニックネームのローカル名を戻します。
get_server_name	ニックネームのサーバー名を戻します。
set_card	ニックネームのカーディナリティーを設定します。
get_card	値が有効な場合は、ニックネームのカーディナリティーを戻します。
set_npages	ニックネームの npages 統計を設定します。

表 85. Nickname_Info クラスのメンバー関数 (続き)

メンバー関数	説明
get_npages	値が有効な場合は、ニックネームの npages 統計を戻します。
set_fpages	ニックネームの fpages 統計を設定します。
get_fpages	値が有効な場合は、ニックネームの fpages 統計を戻します。
set_overflow	ニックネームのオーバーフロー統計を設定します。
get_overflow	値が有効な場合は、ニックネームのオーバーフロー統計を戻します。
insert_column	ニックネーム情報に列定義を追加します。
replace_column	ニックネーム情報の列定義を置換します。
get_number_columns	ニックネームの列数を戻します。
get_first_column	ニックネームの最初の Column_Info オブジェクトへのポインターを戻します。
get_next_column	次の Column_Info オブジェクトへのポインターを戻します。
get_column	列の記述のローカル名を戻します。
merge	デルタ Nickname_Info オブジェクトを現行オブジェクトにマージします。
copy	Nickname_Info オブジェクトを複写します。
add_option	カタログ情報に単一値オプションを追加します。
drop_option	Catalog_Info クラスからオプションを削除します。
get_option	カタログ・オプション名を戻します。
get_first_option	オプション・チェーンの最初のオプションにポインターを戻します。
get_next_option	オプション・チェーンの次のオプションを戻します。

Nickname_Info コンストラクター

目的 デフォルトの (空) Nickname_Info オブジェクトを構成します。

構文

```
Nickname_Info ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 なし。

get_local_schema 関数

目的 ニックネームのローカル・スキーマ名を戻します。

構文

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

入力引き数
なし。

出力引き数

表 86. get_local_schema メンバー関数の出力引き数

名前	データ・タイプ	説明
a_local_schema	sqluint8**	ヌル終了したローカル・スキーマ名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

get_nickname 関数

目的 ニックネームのローカル名を戻します。

構文

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

入力引き数
なし。

出力引き数

表 87. get_nickname メンバー関数の出力引き数

名前	データ・タイプ	説明
a_nickname	sqluint8**	ヌル終了したローカル名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

get_server_name 関数

目的 ニックネームのサーバー名を戻します。

構文

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

入力引き数

なし。

出力引き数

表 88. get_server_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8**	ヌル終了サーバー名へのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_card 関数

目的 ニックネームのカーディナリティーを設定します。

使用法 ラッパーはこのメンバー関数を CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理中に呼び出し、ニックネームの初期カーディナリティーを指定できます。Column_Info の set_colcard メソッドを使用して、各列のカーディナリティーを設定できます。固有の値を持つ列については、ニックネームのカーディナリティーと同じである必要があります。列のカーディナリティーには、ニックネームのカーディナリティーより大きい値を指定できません。

構文

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

入力引き数

表 89. set_card メンバー関数の入力引き数

名前	データ・タイプ	説明
a_card	sqlint64	ニックネームのカーディナリティー。

出力引き数
なし。

戻り値 なし。

get_card 関数

目的 値が有効な場合は、ニックネームのカーディナリティーを戻します。

構文

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

入力引き数
なし。

出力引き数

表 90. get_card メンバー関数の出力引き数

名前	データ・タイプ	説明
a_card	sqlint64*	ニックネームのカーディナリティー。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_npages 関数

目的 ニックネームの npages 統計を設定します。

構文

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

入力引き数

表 91. set_npages メンバー関数の入力引き数

名前	データ・タイプ	説明
a_npages	sqlint32	ニックネームの npages 統計。

出力引き数
なし。

戻り値 なし。

get_npages 関数

目的 値が有効な場合は、ニックネームの npages 統計を戻します。

構文

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

入力引き数
なし。

出力引き数

表 92. get_npages メンバー関数の出力引き数

名前	データ・タイプ	説明
a_npages	sqlint32*	ニックネームの npages 統計。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_fpages 関数

目的 ニックネームの fpages 統計を設定します。

構文

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

入力引き数

表 93. set_fpages メンバー関数の入力引き数

名前	データ・タイプ	説明
a_fpages	sqlint32	ニックネームの fpages 統計。

出力引き数
なし。

戻り値 なし。

get_fpages 関数

目的 値が有効な場合は、ニックネームの fpages 統計を戻します。

構文

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

入力引き数
なし。

出力引き数

表 94. get_fpages メンバー関数の出力引き数

名前	データ・タイプ	説明
a_fpages	sqlint32*	ニックネームの fpages 統計。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

set_overflow 関数

目的 ニックネームのオーバーフロー統計を設定します。

構文

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

入力引き数

表 95. set_overflow メンバー関数の入力引き数

名前	データ・タイプ	説明
a_overflow	sqlint32	ニックネームのオーバーフロー統計。

出力引き数
なし。

戻り値 なし。

get_overflow 関数

目的 値が有効な場合は、ニックネームのオーバーフロー統計を戻します。

構文

```
sqlint32 get_overflow (sqlint32* a_overflow)
```

入力引き数
なし。

出力引き数

表 96. get_overflow メンバー関数の出力引き数

名前	データ・タイプ	説明
a_overflow	sqlint32*	ニックネームのオーバーフロー統計。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は値が設定されていないことを示します。

insert_column 関数

目的 ニックネーム情報に列定義を追加します。

使用法 Column_Info オブジェクトはヒープ上 (新規使用) に割り振る必要があります。このルーチンがポインターを制御します。a_new_col_info 引き数の列 ID が設定されている場合は、エラーが発生します。

構文

```
sqlint32 insert_column (Column_Info* a_new_col_info)
```

入力引き数

表 97. insert_column メンバー関数の入力引き数

名前	データ・タイプ	説明
a_new_col_info	Column_Info*	追加する Column_Info オブジェクト。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

replace_column 関数

目的 ニックネーム情報の列定義を置換します。 2 つの列定義に同じ ID を使用する場合、これらの列定義は同じものとみなされます。

使用法 a_new_col_info 引き数に列 ID が設定されていない場合、エラーが発生します。

構文

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

入力引き数

表 98. replace_column メンバー関数の入力引き数

名前	データ・タイプ	説明
a_new_col_info	Column_Info*	置換する Column_Info。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_number_columns 関数

目的 ニックネームの列数を戻します。

構文

```
sqlint16 get_number_columns ()
```

入力引き数

なし。

出力引き数
なし。

戻り値 列の数。

get_first_column 関数

目的 ニックネームの最初の Column_Info オブジェクトへのポインターを戻します。

使用法 列は ID 順で配列されます。

構文

```
Column_Info* get_first_column ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 最初の Column_Info オブジェクトへのポインター。リストが空の場合は NULL。

get_next_column 関数

目的 次の Column_Info オブジェクトへのポインターを戻します。

使用法 列は ID 順で配列されます。

構文

```
Column_Info* get_next_column (Column_Info* a_cur_col_info)
```

入力引き数

表 99. get_next_column メンバー関数の入力引き数

名前	データ・タイプ	説明
a_cur_col_info	Column_Info*	現行列。

出力引き数
なし。

戻り値 次の Column_Info へのポインター。リストの末尾の場合は NULL。

get_column 関数

目的 列の記述のローカル名を戻します。

構文

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

入力引き数

表 100. get_column メンバー関数の入力引き数

名前	データ・タイプ	説明
a_col_name	sqluint8*	列名 (非ヌル終了)。
a_col_name_len	sqlint32	列名の長さ。

出力引き数

表 101. get_column メンバー関数の出力引き数

名前	データ・タイプ	説明
a_col_info	Column_Info**	Column_Info オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は列名が検出されないことを示します。

merge 関数

目的 デルタ Nickname_Info オブジェクトを現行オブジェクトにマージします。

構文

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

入力引き数

表 102. merge メンバー関数の入力引き数

名前	データ・タイプ	説明
a_delta_info	Nickname_Info*	マージするデータ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

copy 関数

目的 Nickname_Info オブジェクトを複写します。

構文

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

入力引き数

なし。

出力引き数

表 103. copy メンバー関数の出力引き数

名前	データ・タイプ	説明
a_new_idx_info	Nickname_Info**	重複オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

add_option 関数

定義場所

Catalog_Info

目的 カタログ情報に単一値オプションを追加します。

使用法 ラッパーは、CREATE NICKNAME ステートメントまたは ALTER NICKNAME ステートメントの処理中にラッパー生成オプションがカタログに追加された場合、このメンバー関数を呼び出すことができます。

構文

```
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 = "")
```

入力引き数

表 104. add_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_opt_name_len	sqlint32	オプション名の長さ。
a_opt_value	sqluint8*	オプション値 (非ヌル終了)。
a_opt_value_len	sqlint32	オプション値の長さ。
a_act	Catalog_Option::Action	このオプションのアクション (ADD、SET、DROP、または、なし)。

表 104. add_option メンバー関数の入力引き数 (続き)

名前	データ・タイプ	説明
a_opt_type	char*	重複オプションである場合は、SQLN1884 エラー・メッセージに使用されるトークン。sqlqg_misc.h ヘッダー・ファイルで定義された SQLQG_NICKNAME_OPTION 定数を使用します。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

drop_option 関数

定義場所

Catalog_Info

目的 Catalog_Info クラスからオプションを削除します。

使用法 このメンバー関数は、カタログからオプションを削除しません。オプションは、デルタ Catalog_Info オブジェクトに対してオプションが追加された場合に、Catalog_Option::sqlqg_Drop のアクションにより削除されます。

構文

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

入力引き数

表 105. drop_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option*	削除するオプション。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_option 関数

定義場所

Catalog_Info

目的 カタログ・オプション名を戻します。

使用法 オプションが検出されない場合は、出力引き数 a_option が NULL になります。

構文

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

入力引き数

表 106. get_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	オプション名 (非ヌル終了)。
a_name_len	sqlint32	オプション名の長さ。

出力引き数

表 107. get_option メンバー関数の出力引き数

名前	データ・タイプ	説明
a_option	Catalog_Option**	検出されたオプション。

戻り値 戻りコード。0 は成功を示します。SQLQG_NOVALUE は、オプションが検出されなかったことを示します。

get_first_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの最初のオプションにポインターを戻します。

構文

```
Catalog_Option* get_first_option ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 チェーンの最初のオプションへのポインター。チェーンが空の場合は NULL。

get_next_option 関数

定義場所

Catalog_Info

目的 オプション・チェーンの次のオプションを戻します。

構文

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

入力引き数

表 108. get_next_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_current_option	Catalog_Option*	現行オプション。

出力引き数

なし。

戻り値 チェーンの次のオプションへのポインター。チェーンの末尾の場合は NULL。

関連情報:

- 1 ページの『C++ API のカタログ・クラス』

C++ API のラッパー・クラス

以下の表で、C++ API の各ラッパー・クラスについて説明します。

表 109. ラッパー・クラス

クラス名	説明
Unfenced_Generic_Wrapper	このクラスは、unfenced (トラステッド) プロセス・スペース上のラッパーを操作し、カタログ・データを検証します。
Fenced_Generic_Wrapper	このクラスは、fenced プロセス・スペース上のラッパーを操作します。

関連情報:

- 65 ページの『Unfenced_Generic_Wrapper クラス』
- 72 ページの『Fenced_Generic_Wrapper クラス』

Unfenced_Generic_Wrapper クラス

このセクションでは、Unfenced_Generic_Wrapper クラスについて説明し、そのコンストラクター、デストラクター、およびメンバー関数について解説します。

概要

Unfenced_Generic_Wrapper クラスは、unfenced (トラステッド) プロセス・スペース上のラッパーを操作し、カタログ・データを検証します。

Unfenced_Generic_Wrapper クラスは、C++ API のラッパー・クラスの 1 つです。

使用法 ラッパーは、Unfenced_Generic_Wrapper クラスのサブクラスをインプリメントする必要があります。このクラスはラッパー固有の UnfencedWrapper_Hook 関数によりインスタンス化されます。

ファイル

sqlqg_unfenced_generic_wrapper.h

データ・メンバー

以下の表に、Unfenced_Generic_Wrapper クラスとともに使用できるデータ・メンバーをリストします。

表 110. Unfenced_Generic_Wrapper クラスのデータ・メンバー

名前	データ・タイプ	説明
m_info	Wrapper_Info*	ラッパーのカタログ情報を含みます。
m_name	sqluint8*	ラッパー名を含むヌル終了文字ストリングへのポインター。
m_corelib	sqluint8*	ラッパー・ライブラリー名を含むヌル終了文字ストリングへのポインター。名前は、CREATE WRAPPER ステートメントで使用されるラッパー・ライブラリーのものです。この名前は、ステートメントの処理中に使用されるライブラリーには対応しません。
m_version	sqlint32	ラッパー・コードのバージョン。

コンストラクター、デストラクター、およびメンバー関数

以下の表で、Unfenced_Generic_Wrapper クラスのコンストラクター、デストラクター、およびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 111. Unfenced_Generic_Wrapper クラスのコンストラクター

コンストラクター	説明
Unfenced_Generic_Wrapper	Unfenced_Generic_Wrapper クラスのインスタンスを構成します。

表 112. Unfenced_Generic_Wrapper クラスのデストラクター

デストラクター	説明
~Unfenced_Generic_Wrapper	Unfenced_Generic_Wrapper クラスのデストラクター。

表 113. Unfenced_Generic_Wrapper クラスのメンバー関数

メンバー関数	説明
verify_my_register_wrapper_info	CREATE WRAPPER ステートメントがサブミットされた時にカタログ情報を検証します。
verify_my_alter_wrapper_info	ALTER WRAPPER ステートメントがサブミットされた時にカタログ情報を検証します。
get_name	ラッパー名を含むヌル終了文字ストリングにポインターを戻します。
get_corelib	ラッパー・ライブラリー名を含むヌル終了文字ストリングにポインターを戻します。
get_version	ラッパー・コードのバージョンを戻します。
get_info	ラッパー・カタログ情報オブジェクトにポインターを戻します。
initialize_my_wrapper	カタログ情報オブジェクトからのラッパー・オブジェクト状態を初期化します。
create_server	ラッパーに対する適切なサーバーのサブクラスをインスタンス化します。

Unfenced_Generic_Wrapper コンストラクター

目的 Unfenced_Generic_Wrapper のインスタンスを構成します。

構文

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

入力引き数

表 114. Unfenced_Generic_Wrapper コンストラクターの入力引き数

名前	データ・タイプ	説明
a_wrapper_version	sqlint32	ラッパー・コードのバージョン。

出力引き数

表 115. Unfenced_Generic_Wrapper コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りポインター。成功する場合は常に 0。

戻り値 なし。

~Unfenced_Generic_Wrapper デストラクター

目的 Unfenced_Generic_Wrapper クラスのデストラクター。

構文

```
~Unfenced_Generic_Wrapper ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 なし。

verify_my_register_wrapper_info 関数

目的 CREATE WRAPPER ステートメントがサブミットされた時にカタログ情報を検証します。

使用法 ラッパー固有のラッパー・オプションがサポートされる場合、このメンバー関数はラッパーにより、Unfenced_Generic_Wrapper のラッパー固有サブクラスにインプリメントされる場合があります。

ラッパーは、デルタ・オブジェクトを割り振る前に、すでに割り振られていないか検査します。

構文

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

入力引き数

表 116. verify_my_register_wrapper_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_wrapper_info	Wrapper_Info*	CREATE WRAPPER ステートメントからの情報。

出力引き数

表 117. verify_my_register_wrapper_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	Wrapper_Info**	verify_my により追加される情報。

戻り値 戻りコード。0 は成功を示します。

verify_my_alter_wrapper_info 関数

目的 ALTER WRAPPER ステートメントがサブミットされた時にカタログ情報を検証します。

使用法 ラッパー固有のラッパー・オプションがサポートされる場合、このメンバー関数はラッパーにより、Unfenced_Generic_Wrapper のラッパー固有サブクラスにインプリメントされる場合があります。

ラッパーは、デルタ・オブジェクトを割り振る前に、すでに割り振られていないか検査します。

構文

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

入力引き数

表 118. verify_my_alter_wrapper_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_wrapper_info	Wrapper_Info*	ALTER WRAPPER ステートメントからの情報。

出力引き数

表 119. verify_my_alter_wrapper_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	Wrapper_Info**	verify_my により追加される情報。

戻り値 戻りコード。0 は成功を示します。

get_name 関数

定義場所

ラッパー

目的 ラッパー名を含むヌル終了文字ストリングにポインターを戻します。

構文

```
sqluint8* get_name()
```

入力引き数

なし。

出力引き数

なし。

戻り値 文字ストリングへのポインター。

get_corelib 関数

定義場所

ラッパー

目的 ラッパー・ライブラリー名を含むヌル終了文字ストリングにポインターを戻します。名前は、CREATE WRAPPER ステートメントで使用されるラッパー・ライブラリーのものです。この名前は、ステートメントの処理中に使用されるライブラリーには対応しません。

構文

```
sqluint8* get_corelib()
```

入力引き数
なし。

出力引き数
なし。

戻り値 文字ストリングへのポインター。

get_version 関数

定義場所
ラッパー

目的 ラッパー・コードのバージョンを戻します。

構文

```
sqlint32 get_version()
```

入力引き数
なし。

出力引き数
なし。

戻り値 ラッパーのバージョン。

get_info 関数

定義場所
ラッパー

目的 ラッパー・カタログ情報オブジェクトにポインターを戻します。

構文

```
Wrapper_Info* get_info()
```

入力引き数
なし。

出力引き数
なし。

戻り値 カタログ情報オブジェクトへのポインター。

initialize_my_wrapper 関数

定義場所

ラッパー

目的 カタログ情報オブジェクトからのラッパー・オブジェクト状態を初期化します。デフォルトのバージョンでは、この処理は行われません。

使用法 ラッパー固有のラッパー・オプションがサポートされる場合、このメンバー関数は Unfenced_Generic_Wrapper のラッパー固有サブクラスにインプリメントされる場合があります。

構文

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

入力引き数

表 120. initialize_my_wrapper メンバー関数の入力引き数

名前	データ・タイプ	説明
a_wrapper_info	Wrapper_Info*	カタログ情報オブジェクト。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

create_server 関数

定義場所

ラッパー

目的 ラッパーに対する適切なサーバーのサブクラスをインスタンス化します。

使用法 このメンバー関数は、ラッパーにより、Unfenced_Generic_Wrapper のラッパー固有サブクラスにインプリメントされる必要があります。

構文

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

入力引き数

表 121. create_server メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8*	サーバーのヌル終了名。

出力引き数

表 122. create_server メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 新規作成されたサーバー・オブジェクトへのポインター、または NULL。

関連する関数

UnfencedWrapper_Hook 関数は Unfenced_Generic_Wrapper クラスにより使用されませんが、直接このクラスに関連します。

UnfencedWrapper_Hook 関数

目的 DB2 フェデレーテッド (連合)・サーバーが Unfenced_Generic_Wrapper のラッパー固有サブクラスをインスタンス化できるようにするフック関数。この関数はラッパーによりインプリメントし、unfenced ラッパー・モジュールからエクスポートする必要があります。

構文

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

入力引き数
なし。

出力引き数
なし。

戻り値 Unfenced_Generic_Wrapper のインスタンス化されたサブクラス。エラーが発生した場合は NULL。

関連情報:

- 64 ページの『C++ API のラッパー・クラス』

Fenced_Generic_Wrapper クラス

このセクションでは、Fenced_Generic_Wrapper クラスについて説明し、そのコンストラクター、デストラクター、およびメンバー関数について解説します。

概要

Fenced_Generic_Wrapper クラスは、fenced プロセス・スペース上のラッパーを操作します。

Fenced_Generic_Wrapper クラスは、C++ API のラッパー・クラスの 1 つです。

使用法 ラッパーは、Fenced_Generic_Wrapper のサブクラスをインプリメントする必要があります。このクラスはラッパー固有の FencedWrapper_Hook 関数によりインスタンス化されます。

ファイル

sqlqg_fenced_generic_wrapper.h

データ・メンバー

以下の表に、Fenced_Generic_Wrapper クラスとともに使用できるデータ・メンバーをリストします。

表 123. Fenced_Generic_Wrapper クラスのデータ・メンバー

名前	データ・タイプ	説明
m_info	Wrapper_Info*	ラッパーのカタログ情報を含みます。
m_name	sqluint8*	ラッパー名を含む文字ストリングへのポインター。
m_corelib	sqluint8*	ラッパー・ライブラリー名を含む文字ストリングへのポインター。
m_version	sqlint32	ラッパー・コードのバージョン。

コンストラクター、デストラクター、およびメンバー関数

以下の表で、Fenced_Generic_Wrapper クラスのコンストラクター、デストラクター、およびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 124. Fenced_Generic_Wrapper クラスのコンストラクター

コンストラクター	説明
FencedGeneric_Wrapper	Fenced_Generic_Wrapper クラスのインスタンスを構成します。

表 125. Fenced_Generic_Wrapper クラスのデストラクター

デストラクター	説明
~FencedGeneric_Wrapper	Fenced_Generic_Wrapper クラスのデストラクター。

表 126. Fenced_Generic_Wrapper クラスのメンバー関数

メンバー関数	説明
get_name	ラッパー名を含むヌル終了文字ストリングにポインターを戻します。
get_corelib	ラッパー・ライブラリー名を含むヌル終了文字ストリングにポインターを戻します。
get_version	ラッパー・コードのバージョンを戻します。
get_info	ラッパー・カタログ情報オブジェクトにポインターを戻します。
initialize_my_wrapper	カタログ情報オブジェクトからのラッパー・オブジェクト状態を初期化します。
create_server	ラッパーに対する適切なサーバーのサブクラスをインスタンス化します。

FencedGeneric_Wrapper コンストラクター

目的 Fenced_Generic_Wrapper クラスのインスタンスを構成します。

構文

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

入力引き数

表 127. FencedGeneric_Wrapper コンストラクターの入力引き数

名前	データ・タイプ	説明
a_wrapper_version	sqlint32	ラッパー・コードのバージョン。

出力引き数

表 128. FencedGeneric_Wrapper コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。成功する場合は常に 0。

戻り値 なし。

~FencedGeneric_Wrapper デストラクター

目的 Fenced_Generic_Wrapper クラスのデストラクター。

構文

```
~FencedGeneric_Wrapper ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 なし。

get_name 関数

定義場所
ラッパー

目的 ラッパー名を含むヌル終了文字ストリングにポインターを戻します。

構文

```
sqluint8* get_name()
```

入力引き数
なし。

出力引き数
なし。

戻り値 文字ストリングへのポインター。

get_corelib 関数

定義場所
ラッパー

目的 ラッパー・ライブラリー名を含むヌル終了文字ストリングにポインターを戻します。

構文

```
sqluint8* get_corelib()
```

入力引き数
なし。

出力引き数
なし。

戻り値 文字ストリングへのポインター。

get_version 関数

定義場所
ラッパー

目的 ラッパー・コードのバージョンを戻します。

構文

```
sqlint32 get_version()
```

入力引き数
なし。

出力引き数
なし。

戻り値 ラッパーのバージョン。

get_info 関数

定義場所
ラッパー

目的 ラッパー・カタログ情報オブジェクトにポインターを戻します。

構文

```
Wrapper_Info* get_info()
```

入力引き数
なし。

出力引き数
なし。

戻り値 カタログ情報オブジェクトへのポインター。

initialize_my_wrapper 関数

定義場所
ラッパー

目的 カタログ情報オブジェクトからのラッパー・オブジェクト状態を初期化します。デフォルトのバージョンでは、この処理は行われません。

使用法 ラッパー固有のラッパー・オプションがサポートされる場合、ラッパーは Fenced_Generic_Wrapper のラッパー固有サブクラスをインプリメントできます。

構文

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

入力引き数

表 129. initialize_my_wrapper メンバー関数の入力引き数

名前	データ・タイプ	説明
a_wrapper_info	Wrapper_Info*	カタログ情報オブジェクト。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

create_server 関数

定義場所

ラッパー

目的 ラッパーに対する適切なサーバーのサブクラスをインスタンス化します。

使用法 ラッパーは、Fenced_Generic_Wrapper のラッパー固有サブクラスに、このメンバー関数をオーバーライドする必要があります。

構文

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

入力引き数

表 130. create_server メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8*	サーバーのヌル終了名。

出力引き数

表 131. create_server メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 新規作成されたサーバー・オブジェクトへのポインター、または NULL。

関連する関数

FencedWrapper_Hook 関数は Fenced_Generic_Wrapper クラスにより使用されませんが、直接このクラスに関連します。

FencedWrapper_Hook 関数

目的 DB2 フェデレーテッド (連合)・サーバーが Fenced_Generic_Wrapper のラッパー固有サブクラスをインスタンス化できるようにするフック関数。この関数はラッパーによりインプリメントし、fenced ラッパー・モジュールからエクスポートする必要があります。

構文

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

入力引き数
なし。

出力引き数
なし。

戻り値 Fenced_Generic_Wrapper のインスタンス化されたサブクラス。エラーが発生した場合は NULL。

関連情報:

- 64 ページの『C++ API のラッパー・クラス』

C++ API のサーバー・クラス

以下の表で、C++ API の各サーバー・クラスについて説明します。

表 132. サーバー・クラス

クラス名	説明
Unfenced_Generic_Server	Unfenced_Generic_Server クラスは Unfenced_Server クラスのサブクラスであり、unfenced (トラステッド) プロセス・スペースで操作されるすべてのサーバー機能の抽象ベース・クラスです。Unfenced_Generic_Server クラスは、CREATE SERVER ステートメントおよび ALTER SERVER ステートメントからのカタログ情報を検証します。

表 132. サーバー・クラス (続き)

クラス名	説明
Fenced_Generic_Server	Fenced_Generic_Server クラスは Server クラスのサブクラスであり、fenced (非トラステッド) プロセス・スペースで操作されるすべてのサーバー機能の抽象ベース・クラスです。Fenced_Generic_Server クラスは、リモート接続およびニックネームを作成します。

関連情報:

- 79 ページの『Unfenced_Generic_Server クラス』
- 90 ページの『Fenced_Generic_Server クラス』

Unfenced_Generic_Server クラス

このセクションでは、Unfenced_Generic_Server クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Unfenced_Generic_Server クラスは Unfenced_Server クラスのサブクラスであり、unfenced (トラステッド) プロセス・スペースで操作されるすべてのサーバー機能の抽象ベース・クラスです。このクラスは、CREATE SERVER ステートメントおよび ALTER SERVER ステートメントからのカタログ情報を検証します。

Unfenced_Generic_Server クラスは、C++ API のサーバー・クラスの 1 つです。

使用法 ラッパーは、Unfenced_Generic_Wrapper のサブクラスをインプリメントする必要があります。 Unfenced_Generic_Server クラスは、ラッパーにより Unfenced_Generic_Wrapper のラッパー固有サブクラスの create_server() メソッドにインスタンス化されます。

ファイル

sqlqg_unfenced_generic_server.h

データ・メンバー

以下の表に、Unfenced_Generic_Server クラスとともに使用できるデータ・メンバーをリストします。

表 133. Unfenced_Generic_Server クラスのデータ・メンバー

名前	データ・タイプ	説明
info	Server_Info*	サーバーのカタログ情報。

表 133. Unfenced_Generic_Server クラスのデータ・メンバー (続き)

名前	データ・タイプ	説明
name	sqluint8*	サーバー名を含むヌル終了文字ストリング。
kind	server_kind	サーバーの種類。常に Server::generic_kind。
wrapper	Wrapper*	このサーバーを所有する関連ラッパー・オブジェクトへのポインター。

コンストラクターおよびメンバー関数

以下の表で、Unfenced_Generic_Server クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 134. Unfenced_Generic_Server クラスのコンストラクター

コンストラクター	説明
Unfenced_Generic_server	Unfenced_Generic_Server クラスのインスタンスを構成します。

表 135. Unfenced_Generic_Server クラスのメンバー関数

メンバー関数	説明
create_remote_user	Remote_User の適切なサブクラスをインスタンス化します。
plan_request	提案されたプランを分析し、どの程度の要求 (存在する場合) をデータ・ソースに送るか決定します。
get_selectivity	述部リストの選択性を見積もります。
create_reply	応答オブジェクトをインスタンス化します。
get_name	ヌル終了サーバー名にポインターを戻します。
get_type	カタログ情報から CREATE SERVER ステートメントに指定されたヌル終了サーバー・タイプを戻します。
get_version	CREATE SERVER ステートメントに指定されたヌル終了サーバー・バージョンを戻します。

表 135. Unfenced_Generic_Server クラスのメンバー関数 (続き)

メンバー関数	説明
get_info	保管カタログ情報オブジェクトにポインターを戻します。
initialize_my_server	有効なカタログ情報からのサーバー・インスタンスを初期化します。
create_nickname	このサーバーに対する適切なニックネームのサブクラスをインスタンス化します。
is_reserved_server_option	DB2 にサポートされ、ラッパーにより無視されるオプションを識別します。
verify_my_register_server_info	CREATE SERVER ステートメントで提供される情報を検証します。
verify_my_alter_server_info	ALTER SERVER ステートメントで提供される情報を検証します。

Unfenced_Generic_server コンストラクター

目的 Unfenced_Generic_Server クラスのインスタンスを構成します。

構文

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

入力引き数

表 136. Unfenced_Generic_server コンストラクターの入力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8*	サーバー名を含むヌル終了ストリング。
a_wrapper	UnfencedWrapper*	このサーバー・オブジェクトを所有するラッパー・オブジェクトへのポインター。

出力引き数

表 137. Unfenced_Generic_server コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 なし。

create_remote_user 関数

目的 Remote_User の適切なサブクラスをインスタンス化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる場合があります。 Unfenced_Generic_User のラッパー固有サブクラスがインプリメントされている場合、このメンバー関数をインプリメントする必要があります。

構文

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

入力引き数

表 138. create_remote_user メンバー関数の入力引き数

名前	データ・タイプ	説明
a_user_name	sqluint8*	ユーザー名を含むヌル終了ストリング。

出力引き数

表 139. create_remote_user メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 Remote_User オブジェクトへのポインター。

plan_request 関数

目的 提案されたプランを分析し、どの程度の要求 (存在する場合) をデータ・ソースに送るか決定します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる必要があります。

構文

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

入力引き数

表 140. plan_request メンバー関数の入力引き数

名前	データ・タイプ	説明
a_req	Request*	提案されたプランを記述する要求オブジェクトへのポインター。

出力引き数

表 141. plan_request メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rpl	Reply**	plan_request により構成される応答オブジェクトのチェーンへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_selectivity 関数

目的 述部リストの選択性を見積もります。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる場合があります。デフォルト・バージョンでは、標準装備の DB2 Information Integrator 公式を使用して選択性を見積もります。ラッパーはこれらの公式と、述部式間の既知の従属関係を使用します。

構文

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

入力引き数

表 142. get_selectivity メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pl	Predicate_List*	述部のリスト。

出力引き数

表 143. get_selectivity メンバー関数の出力引き数

名前	データ・タイプ	説明
a_selectivity	float*	0.0 から 1.0 の値で表される選択性。

戻り値 戻りコード。0 は成功を示します。

create_reply 関数

目的 応答オブジェクトをインスタンス化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる場合があります。ラッパー固有の Reply サブクラスを使用する場合は必ずインプリメントする必要があります。ラッパーが応答をサブクラス化して自身のコスト・モデルをインプリメントした場合、このメソッドはオーバーライドされます。

構文

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

入力引き数

表 144. create_reply メンバー関数の入力引き数

名前	データ・タイプ	説明
a_req	Request*	要求オブジェクトへのポインター。

出力引き数

表 145. create_reply メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rpl	Reply**	応答オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_name 関数

定義場所

サーバー

目的 ヌル終了サーバー名にポインターを戻します。

構文

```
sqluint8* get_name()
```

入力引き数
なし。

出力引き数
なし。

戻り値 サーバー名へのポインター。

get_type 関数

定義場所
サーバー

目的 カタログ情報から CREATE SERVER ステートメントに指定されたヌル終了サーバー・タイプを戻します。

使用法 このメソッドは、サーバー・オブジェクトが初期化された後 にのみ有効になります。カタログにサーバーのタイプが指定されていない場合、このメソッドはエラーをログに記録します。ただし、ユーザー・エラーは生成されません。

構文

```
sqluint8* get_type()
```

入力引き数
なし。

出力引き数
なし。

戻り値 サーバー・タイプへのポインター。

get_version 関数

定義場所
サーバー

目的 CREATE SERVER ステートメントに指定されたヌル終了サーバー・バージョンを戻します。

使用法 このメソッドは、サーバー・オブジェクトが初期化された後にのみ有効になります。カタログにサーバーのタイプが指定されていない場合、このメソッドはエラーをログに記録します。ただし、ユーザー・エラーは生成されません。

構文

```
sqluint8* get_version()
```

入力引き数
なし。

出力引き数
なし。

戻り値 サーバー・バージョンへのポインター。

get_info 関数

定義場所
サーバー

目的 保管カタログ情報オブジェクトにポインターを戻します。

使用法 このメソッドは、サーバーの初期化後にのみ有効になります。

構文

```
Server_Info* get_info()
```

入力引き数
なし。

出力引き数
なし。

戻り値 カタログ情報オブジェクトへのポインター。

initialize_my_server 関数

定義場所
サーバー

目的 有効なカタログ情報からのサーバー・インスタンスを初期化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる場合があります。ラッパー固有のサーバー・オプションがサポートされる場合は必ずインプリメントする必要があります。

構文

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

入力引き数

表 146. initialize_my_server メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_info	Server_Info*	検証されたカタログ情報。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

create_nickname 関数

定義場所

サーバー

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる必要があります。

目的 このサーバーに対する適切なニックネームのサブクラスをインスタンス化します。

構文

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

入力引き数

表 147. create_nickname メンバー関数の入力引き数

名前	データ・タイプ	説明
a_schema_name	sqluint8*	ニックネームのローカル (DB2) スキーマ名を含むヌル終了ストリング。
a_nickname_name	sqluint8*	ローカル (DB2) ニックネーム名を含むヌル終了ストリング。

出力引き数

表 148. create_nickname メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 インスタンス化されたニックネーム・オブジェクトへのポインター。

is_reserved_server_option 関数

定義場所

UnfencedServer

目的 DB2 にサポートされ、ラッパーにより無視されるオプションを識別します。

構文

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

入力引き数

表 149. is_reserved_server_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_op_name	sqluint8*	オプション名を含むヌル終了ストリング。

出力引き数

なし。

戻り値 0 は、そのオプションが DB2 で定義されたサーバー・オプションではないことを示します。0 以外の場合は、そのオプションが DB2 で定義済みのサーバー・オプションであることを示します。

verify_my_register_server_info 関数

定義場所

UnfencedServer

目的 CREATE SERVER ステートメントで提供される情報を検証します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる場合があります。ラッパー固有のサーバー・オプションがサポートされる場合は必ずインプリメントする必要があります。ラッパーは、a_delta_info オブジェクトを割り振る前に、それが割り振り済みかどうか検査する必要があります。

構文

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

入力引き数

表 150. verify_my_register_server_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_info	Server_Info*	CREATE SERVER ステートメントからの情報。

出力引き数

表 151. verify_my_register_server_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	Server_Info**	ラッパーが提供する追加情報。

戻り値 戻りコード。0 は成功を示します。

verify_my_alter_server_info 関数

定義場所

UnfencedServer

目的 ALTER SERVER ステートメントで提供される情報を検証します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced server サブクラスにインプリメントされる場合があります。ラッパー固有のサーバー・オプションがサポートされる場合は必ずインプリメントする必要があります。ラッパーは、a_delta_info オブジェクトを割り振る前に、それが割り振り済みかどうか検査する必要があります。

構文

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

入力引き数

表 152. verify_my_alter_server_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_info	Server_Info*	ALTER SERVER ステートメントからの情報。

出力引き数

表 153. verify_my_alter_server_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	Server_Info**	ラッパーが提供する追加情報。

戻り値 戻りコード。0 は成功を示します。

関連情報:

• 78 ページの『C++ API のサーバー・クラス』

Fenced_Generic_Server クラス

このセクションでは、Fenced_Generic_Server クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Fenced_Generic_Server クラスは Server クラスのサブクラスであり、fenced (非トラステッド) プロセス・スペースで操作されるすべてのサーバー機能の抽象ベース・クラスです。このクラスは、リモート接続およびニックネームを作成します。

Fenced_Generic_Server クラスは、C++ API のサーバー・クラスの 1 つです。

使用法 ラッパーは、Fenced_Generic_Wrapper のサブクラスをインプリメントする必要があります。このクラスはラッパーにより、Fenced_Generic_Wrapper のラッパー固有サブクラスの create_server() メソッドにインスタンス化されます。

ファイル

sqlqg_fenced_genserver.h

データ・メンバー

以下の表に、Fenced_Generic_Server クラスとともに使用できるデータ・メンバーをリストします。

表 154. Fenced_Generic_Server クラスのデータ・メンバー

名前	データ・タイプ	説明
info	Server_Info*	サーバーのカタログ情報。
name	sqluint8*	サーバー名を含むヌル終了文字ストリング。
kind	server_kind	サーバーの種類。常に Server::generic_kind。
wrapper	Wrapper*	このサーバーを所有する関連ラッパー・オブジェクトへのポインター。

コンストラクターおよびメンバー関数

以下の表で、Fenced_Generic_Server クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 155. Fenced_Generic_Server クラスのコンストラクター

コンストラクター	説明
Fenced_Generic_Server	このクラスのコンストラクター。

表 156. Fenced_Generic_Server クラスのメンバー関数

メンバー関数	説明
create_remote_connection	Remote_Connection の適切なサブクラスをインスタンス化します。
get_name	ヌル終了サーバー名にポインターを戻します。
get_type	カタログ情報から CREATE SERVER ステートメントに指定されたヌル終了サーバー・タイプを戻します。
get_version	CREATE SERVER ステートメントに指定されたヌル終了サーバー・バージョンを戻します。
get_info	保管カタログ情報オブジェクトにポインターを戻します。
initialize_my_server	有効なカタログ情報からのサーバー・インスタンスを初期化します。
create_remote_user	このラッパーに対する Remote_User の適切なサブクラスをインスタンス化します。
create_nickname	このサーバーに対する適切なニックネームのサブクラスをインスタンス化します。

Fenced_Generic_Server コンストラクター

目的 このクラスのコンストラクター。

構文

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

入力引き数

表 157. Fenced_Generic_Server コンストラクターの入力引き数

名前	データ・タイプ	説明
a_server_name	sqluint8*	サーバーの名前。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

create_remote_connection 関数

定義場所
FencedServer

目的 Remote_Connection の適切なサブクラスをインスタンス化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の fenced server サブクラスにインプリメントされる必要があります。

構文

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

入力引き数

表 158. create_remote_connection メンバー関数の入力引き数

名前	データ・タイプ	説明
a_user	FencedRemote_User*	接続に使用するユーザーを表すオブジェクトへのポインター。

出力引き数

表 159. create_remote_connection メンバー関数の出力引き数

名前	データ・タイプ	説明
a_connection	Remote_Connection**	リモート接続オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_name 関数

定義場所
サーバー

目的 ヌル終了サーバー名にポインターを戻します。

構文

```
sqluint8* get_name()
```

入力引き数
なし。

出力引き数
なし。

戻り値 サーバー名へのポインター。

get_type 関数

定義場所
サーバー

目的 カタログ情報から CREATE SERVER ステートメントに指定されたヌル終了サーバー・タイプを戻します。

使用法 このメソッドは、サーバー・オブジェクトが初期化された後 (つまり initialize_server の実行後) にのみ有効になります。カタログにサーバーのタイプが指定されていない場合は、このメソッドを呼び出すことでエラーがログに記録されます。ただし、ユーザー・エラーは生成されません。

構文

```
sqluint8* get_type()
```

入力引き数
なし。

出力引き数
なし。

戻り値 サーバー・タイプへのポインター。

get_version 関数

定義場所
サーバー

目的 CREATE SERVER ステートメントに指定されたヌル終了サーバー・バージョンを戻します。

使用法 このメソッドは、サーバー・オブジェクトが初期化された後 (つまり initialize_server の実行後) にのみ有効になります。カタログにサーバーのタイプが指定されていない場合は、このメソッドを呼び出すことでエラーがログに記録されます。ただし、ユーザー・エラーは生成されません。

構文

```
sqluint8* get_version()
```

入力引き数
なし。

出力引き数
なし。

戻り値 サーバー・バージョンへのポインター。

get_info 関数

定義場所
サーバー

目的 保管カタログ情報オブジェクトにポインターを戻します。

使用法 このメソッドは、サーバーの初期化後にのみ有効になります。

構文

```
Server_Info* get_info()
```

入力引き数
なし。

出力引き数
なし。

戻り値 カタログ情報オブジェクトへのポインター。

initialize_my_server 関数

定義場所
サーバー

目的 有効なカタログ情報からのサーバー・インスタンスを初期化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の fenced server サブクラスにインプリメントされる場合があります。ラッパー固有のサーバー・オプションがサポートされる場合は、このメンバー関数をインプリメントする必要があります。

構文

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

入力引き数

表 160. initialize_my_server メンバー関数の入力引き数

名前	データ・タイプ	説明
a_server_info	Server_Info*	検証されたカタログ情報。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

create_remote_user 関数

定義場所

サーバー

目的 このラッパーに対する Remote_User の適切なサブクラスをインスタンス化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の fenced server サブクラスにインプリメントされる場合があります。ラッパー固有の Fenced_Generic_User サブクラスを使用する場合は、このメンバー関数をインプリメントする必要があります。

構文

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

入力引き数

表 161. create_remote_user メンバー関数の入力引き数

名前	データ・タイプ	説明
a_user_name	sqluint8*	リモート・ユーザーの名前。

出力引き数

表 162. create_remote_user メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 インスタンス化された Remote_User へのポインター。

create_nickname 関数

定義場所

サーバー

目的 このサーバーに対する適切なニックネームのサブクラスをインスタンス化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の fenced server サブクラスにインプリメントされる必要があります。

構文

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

入力引き数

表 163. create_nickname メンバー関数の入力引き数

名前	データ・タイプ	説明
a_schema_name	sqluint8*	ニックネームのローカル (DB2) スキーマ名を含むヌル終了ストリング。
a_nickname_name	sqluint8*	ローカル (DB2) ニックネーム名を含むヌル終了ストリング。

出力引き数

表 164. create_nickname メンバー関数の出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 インスタンス化されたニックネーム・オブジェクトへのポインター。

関連情報:

- 78 ページの『C++ API のサーバー・クラス』

C++ API のユーザー・クラス

以下の表で、C++ API の各ユーザー・クラスについて説明します。

表 165. ユーザー・クラス

クラス名	説明
Unfenced_Generic_User	このクラスは、unfenced (トラステッド) プロセス・スペース内のユーザー・マッピングを扱います。このクラスは、CREATE USER MAPPING ステートメントおよび ALTER USER MAPPING ステートメントからの情報を検証します。
Fenced_Generic_User	このクラスは、fenced (非トラステッド) プロセス・スペース内のユーザー・マッピングを扱います。

関連情報:

- 97 ページの『Unfenced_Generic_User クラス』
- 103 ページの『Fenced_Generic_User クラス』

Unfenced_Generic_User クラス

このセクションでは、Unfenced_Generic_User クラスについて説明し、そのコンストラクター、デストラクター、およびメンバー関数について解説します。

概要

Unfenced_Generic_User クラスは、unfenced (トラステッド) プロセス・スペース内のユーザー・マッピングを扱います。このクラスは、CREATE USER MAPPING ステートメントおよび ALTER USER MAPPING ステートメントからの情報を検証します。

Unfenced_Generic_User クラスは、C++ API のユーザー・クラスの 1 つです。

使用法 CREATE USER MAPPING ステートメントまたは ALTER USER MAPPING ステートメントのラッパー固有オプションを使用する場合、ラッパーは Unfenced_Generic_User のサブクラスをインプリメントする必要があります。このクラスは、オプションの検証にのみ使用され、ラッパーにより Unfenced_Generic_Server のラッパー固有サブクラスの create_remote_user() メソッドにインスタンス化されます。

ファイル

sqlqg_unfenced_generic_user.h

データ・メンバー

以下の表に、Unfenced_Generic_User クラスとともに使用できるデータ・メンバーをリストします。

表 166. Unfenced_Generic_User クラスのデータ・メンバー

名前	データ・タイプ	説明
local_name	sqluint8*	ヌル終了ユーザー名 (ローカル)。
server	Server*	このユーザーを所有するサーバー・オブジェクトへのポインター。
info	User_Info*	カタログ情報のローカル・コピー。このデータ・メンバーは、initialize_my_user の後有効になります。

コンストラクター、デストラクター、およびメンバー関数

以下の表で、Unfenced_Generic_User クラスのコンストラクター、デストラクター、およびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 167. Unfenced_Generic_User クラスのコンストラクター

コンストラクター	説明
Unfenced_Generic_User	Unfenced_Generic_User のインスタンスを構成します。

表 168. Unfenced_Generic_User クラスのデストラクター

デストラクター	説明
~Unfenced_Generic_User	Unfenced_Generic_User のインスタンスを破棄します。

表 169. Unfenced_Generic_User クラスのメンバー関数

メンバー関数	説明
get_info	User_Info のローカル・コピーにポインターを戻します。
is_reserved_user_option	指定されたカタログ・オプションが、DB2 で予約済みのユーザー・マッピングのオプションであるか検証します。
initialize_my_user	有効なカタログ情報からのオブジェクトの状態を初期化します。

表 169. Unfenced_Generic_User クラスのメンバー関数 (続き)

メンバー関数	説明
verify_my_register_user_info	CREATE USER MAPPING ステートメントからの情報を検証します。
verify_my_alter_user_info	ALTER USER MAPPING ステートメントからの情報を検証します。

Unfenced_Generic_User コンストラクター

目的 Unfenced_Generic_User のインスタンスを構成します。

構文

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

入力引き数

表 170. Unfenced_Generic_User コンストラクターの入力引き数

名前	データ・タイプ	説明
a_user_name	sqluint8*	ヌル終了ユーザー名 (ローカル)。
a_server	UnfencedServer*	このユーザーを所有するサーバー・オブジェクトへのポインター。

出力引き数

表 171. Unfenced_Generic_User コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	このユーザーを所有するサーバー・オブジェクトへのポインター。

戻り値 なし。

~Unfenced_Generic_User デストラクター

目的 Unfenced_Generic_User のインスタンスを破棄します。

構文

```
virtual ~Unfenced_Generic_User ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 なし。

get_info 関数

目的 User_Info のローカル・コピーにポインターを戻します。

構文

```
User_Info* get_info ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 User_Info オブジェクトへのポインター。

is_reserved_user_option 関数

目的 指定されたカタログ・オプションが、DB2 で予約済みのユーザー・マッピングのオプションであるか検証します。

構文

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

入力引き数

表 172. is_reserved_user_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	チェック対象となるヌル終了したオプション名。

出力引き数
なし。

戻り値 0 は、そのオプションが DB2 の予約済みオプションではないことを示します。

initialize_my_user 関数

目的 有効なカタログ情報からのオブジェクトの状態を初期化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced user サブクラスにインプリメントされる場合があります。

構文

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

入力引き数

表 173. initialize_my_user メンバー関数の入力引き数

名前	データ・タイプ	説明
a_user_info	User_Info*	検証されたカタログ情報。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

verify_my_register_user_info 関数

目的 CREATE USER MAPPING ステートメントからの情報を検証します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced user サブクラスにインプリメントされる場合があります。ラッパー固有のユーザー・マッピング・オプションがサポートされる場合は、このメンバー関数をインプリメントする必要があります。

ラッパーは、a_delta_info オブジェクトを割り振る前に、すでに割り振られていないか検査します。

構文

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

入力引き数

表 174. verify_my_register_user_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_user_info	User_Info*	CREATE USER MAPPING ステートメントからの情報。

出力引き数

表 175. verify_my_register_user_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	User_Info**	ラッパーが提供する追加情報。

戻り値 戻りコード。0 は成功を示します。

verify_my_alter_user_info 関数

目的 ALTER USER MAPPING ステートメントからの情報を検証します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced user サブクラスにインプリメントされる場合があります。ラッパー固有のユーザー・マッピング・オプションがサポートされる場合は、このメンバー関数をインプリメントする必要があります。

ラッパーは、a_delta_info オブジェクトを割り振る前に、すでに割り振られていないか検査します。

構文

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

入力引き数

表 176. verify_my_alter_user_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_user_info	User_Info*	ALTER USER MAPPING ステートメントからの情報。

出力引き数

表 177. verify_my_alter_user_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	User_Info**	ラッパーが提供する追加情報。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 97 ページの『C++ API のユーザー・クラス』

Fenced_Generic_User クラス

このセクションでは、Fenced_Generic_User クラスについて説明し、そのコンストラクター、デストラクター、およびメンバー関数について解説します。

概要

Fenced_Generic_User クラスは、fenced (非トラステッド) プロセス・スペース内のユーザー・マッピングを扱います。

Fenced_Generic_User クラスは、C++ API のユーザー・クラスの 1 つです。

使用法 このクラスは、ラッパーにより Fenced_Generic_Server のラッパー固有サブクラスの create_remote_user() メソッドにインスタンス化されます。ラッパー固有のユーザー・マッピング・オプションがサポートされる場合、ラッパーは Fenced_Generic_User のサブクラスをインプリメントします。

ファイル

sqlqg_fenced_generic_user.h

データ・メンバー

以下の表に、Fenced_Generic_User クラスとともに使用できるデータ・メンバーをリストします。

表 178. Fenced_Generic_User クラスのデータ・メンバー

名前	データ・タイプ	説明
local_name	sqluint8*	ヌル終了ユーザー名 (ローカル)。
server	Server*	このユーザーを所有するサーバー・オブジェクトへのポインター。
info	User_Info*	カタログ情報のローカル・コピー。このデータ・メンバーは、initialize_my_user の後有効になります。

コンストラクター、デストラクター、およびメンバー関数

以下の表で、Fenced_Generic_User クラスのコンストラクター、デストラクター、およびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 179. Fenced_Generic_User クラスのコンストラクター

コンストラクター	説明
Fenced_Generic_User	Fenced_Generic_User のインスタンスを構成します。

表 180. Fenced_Generic_User クラスのデストラクター

デストラクター	説明
~Fenced_Generic_User	Fenced_Generic_User のインスタンスを破棄します。

表 181. Fenced_Generic_User クラスのメンバー関数

メンバー関数	説明
get_info	User_Info のローカル・コピーにポインターを戻します。
initialize_my_user	有効なカタログ情報からのオブジェクトの状態を初期化します。

Fenced_Generic_User コンストラクター

目的 Fenced_Generic_User のインスタンスを構成します。

構文

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

入力引き数

表 182. Fenced_Generic_User コンストラクターの入力引き数

名前	データ・タイプ	説明
a_local_user_name	sqluint8*	ヌル終了ユーザー名 (ローカル)。
a_server	FencedServer*	このユーザーを所有するサーバー・オブジェクトへのポインター。

出力引き数

表 183. Fenced_Generic_User コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 なし。

~Fenced_Generic_User デストラクター

目的 Fenced_Generic_User のインスタンスを破棄します。

構文

```
virtual ~Fenced_Generic_User ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 なし。

get_info 関数

目的 User_Info のローカル・コピーにポインターを戻します。

構文

```
User_Info* get_info ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 User_Info オブジェクトへのポインター。

initialize_my_user 関数

目的 有効なカタログ情報からのオブジェクトの状態を初期化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の fenced user サブクラ

スにインプリメントされる場合があります。ラッパー固有のユーザー・マッピング・オプションがサポートされる場合は必ずインプリメントする必要があります。

構文

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

入力引き数

表 184. initialize_my_user メンバー関数の入力引き数

名前	データ・タイプ	説明
a_user_info	User_Info*	検証されたカタログ情報。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 97 ページの『C++ API のユーザー・クラス』

C++ API のニックネーム・クラス

以下の表で、C++ API の各ニックネーム・クラスについて説明します。

表 185. ニックネーム・クラス

クラス名	説明
Unfenced_Generic_Nickname	このクラスは、unfenced (トラステッド) プロセス・スペース内のニックネームを処理します。このクラスは、CREATE NICKNAME ステートメントおよび ALTER NICKNAME ステートメントからの情報を検証します。
Fenced_Generic_Nickname	このクラスは、fenced (非トラステッド) プロセス・スペース内のニックネームを処理します。このクラスは、CREATE NICKNAME ステートメントからの情報を検証します。

関連情報:

- 107 ページの『Unfenced_Generic_Nickname クラス』
- 115 ページの『Fenced_Generic_Nickname クラス』

Unfenced_Generic_Nickname クラス

このセクションでは、Unfenced_Generic_Nickname クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Unfenced_Generic_Nickname クラスは、unfenced (トラステッド) プロセス・スペース内のニックネームを処理します。このクラスは、CREATE NICKNAME ステートメントおよび ALTER NICKNAME ステートメントからの情報を検証します。

Unfenced_Generic_Nickname クラスは、C++ API のニックネーム・クラスの 1 つです。

使用法 このクラスは、ラッパーによりサブクラス化される必要があります。このクラスは、Unfenced_Generic_Server のラッパー固有サブクラスの create_nickname() メソッドにより、ラッパーでインスタンス化されます。

ファイル

sqlqg_unfenced_generic_nickhame.h

データ・メンバー

以下の表に、Unfenced_Generic_Nickname クラスとともに使用できるデータ・メンバーをリストします。

表 186. Unfenced_Generic_Nickname クラスのデータ・メンバー

名前	データ・タイプ	説明
name	sqluint8*	ニックネームの名前 (ローカル) を含むヌル終了ストリング。
schema	sqluint8*	スキーマの名前 (ローカル) を含むヌル終了ストリング。
server	Server*	このニックネームを所有するサーバー・オブジェクトへのポインター。
m_cardinality	sqlint64	カーディナリティー。
m_advance_cost	sqlint32	行を取り出すコスト (MS (ミリ秒))。
m_setup_cost	sqlint32	照会の 1 回限りのセットアップを実行するコスト (MS (ミリ秒))。

表 186. Unfenced_Generic_Nickname クラスのデータ・メンバー (続き)

名前	データ・タイプ	説明
m_submission_cost	sqlint32	照会をサブミットするコスト (MS (ミリ秒)) (1 回限りのセットアップ・コストを除く)。

コンストラクターおよびメンバー関数

以下の表で、Unfenced_Generic_Nickname クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 187. Unfenced_Generic_Nickname クラスのコンストラクター

コンストラクター	説明
Unfenced_Generic_Nickname	Unfenced_Generic_Nickname のインスタンスを構成します。

表 188. Unfenced_Generic_Nickname クラスのメンバー関数

メンバー関数	説明
get_local_name	ヌル終了したローカル・ニックネーム名へポインターを戻します。
get_local_schema	ヌル終了したローカル・スキーマ名へポインターを戻します。
get_server	収容サーバー・オブジェクトへポインターを戻します。
is_reserved_nickname_option	特定のオプション名が DB2 に組み込まれたニックネーム・オプションの 1 つであるかどうか識別します。
is_reserved_column_option	特定のオプション名が DB2 に組み込まれた列オプションの 1 つであるかどうか識別します。
initialize_my_nickname	カタログからの有効な情報を使用してニックネームを初期化します。
verify_my_register_nickname_info	CREATE NICKNAME ステートメントからの情報を検証します。
verify_my_alter_nickname_info	ALTER NICKNAME ステートメントからの情報を検証します。
get_card	ニックネームのカーディナリティーを戻します。

表 188. Unfenced_Generic_Nickname クラスのメンバー関数 (続き)

メンバー関数	説明
get_advance_cost	デフォルトのコスト・モデルで使用される ADVANCE_COST ニックネーム・オプションの値を戻します。
get_setup_cost	デフォルトのコスト・モデルで使用される SETUP_COST ニックネーム・オプションの値を戻します。
get_submission_cost	デフォルトのコスト・モデルで使用される SUBMISSION_COST ニックネーム・オプションの値を戻します。

Unfenced_Generic_Nickname コンストラクター

目的 Unfenced_Generic_Nickname のインスタンスを構成します。

構文

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

入力引き数

表 189. Unfenced_Generic_Nickname コンストラクターの入力引き数

名前	データ・タイプ	説明
a_schema	sqluint8*	ヌル終了スキーマ名。
a_nickname_name	sqluint8*	ヌル終了ニックネーム名 (ローカル)。
a_nickname_server	Unfenced_Generic_Server*	収容サーバー・オブジェクトへのポインター。

出力引き数

表 190. Unfenced_Generic_Nickname コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 なし。

get_local_name 関数

目的 ヌル終了したローカル・ニックネーム名へポインターを戻します。

構文

```
sqluint8* get_local_name ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 ヌル終了ニックネーム名。

get_local_schema 関数

目的 ヌル終了したローカル・スキーマ名へポインターを戻します。

構文

```
sqluint8* get_local_schema ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 ヌル終了スキーマ名。

get_server 関数

目的 収容サーバー・オブジェクトへポインターを戻します。

構文

```
Server* get_server ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 収容サーバー・オブジェクトへのポインター。

is_reserved_nickname_option 関数

目的 特定のオプション名が DB2 に組み込まれたニックネーム・オプションの 1 つであるかどうか識別します。

構文

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

入力引き数

表 191. is_reserved_nickname_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	ヌル終了したオプション名ストリング。

出力引き数

なし。

戻り値 0 は、そのオプションが予約済みニックネーム・オプションではないことを示します。0 以外の場合は、そのオプションが予約済みニックネーム・オプションであることを示します。

is_reserved_column_option 関数

目的 特定のオプション名が DB2 に組み込まれた列オプションの 1 つであるかどうか識別します。

構文

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

入力引き数

表 192. is_reserved_column_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	ヌル終了したオプション名ストリング。

出力引き数

なし。

戻り値 0 は、そのオプションが予約済みオプションではないことを示します。0 以外の場合は、そのオプションが予約済み列オプションであることを示します。

initialize_my_nickname 関数

目的 カタログからの有効な情報を使用してニックネームを初期化します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の unfenced nickname サブクラスにインプリメントされる場合があります。

構文

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

入力引き数

表 193. initialize_my_nickname メンバー関数の入力引き数

名前	データ・タイプ	説明
a_cat_info	Nickname_Info*	カタログからの情報。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

verify_my_register_nickname_info 関数

目的 CREATE NICKNAME ステートメントからの情報を検証します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の fenced nickname サブクラスにインプリメントされる場合があります。ラッパー固有のニックネーム・オプションまたは列オプションがサポートされる場合は、このメソッドまたは隔離クラスの同じメソッドをインプリメントする必要があります。

verify_my_register_nickname_info 関数はトラステッド・プロセス・スペースの一部であるため、このメンバー関数はリモート・データ・ソースと対話できません。ニックネーム情報の検証に対話が必要な場合は、Fenced_Generic_Nickname クラスの verify_my_register_nickname_info メンバー関数をインプリメントする必要があります。

ラッパーは、a_delta_info オブジェクトを割り振る前に、すでに割り振られていないか検査します。このメソッドは、Fenced_Generic_Nickname::verify_my_register_nickname_info が呼び出されてから呼び出されます。

構文

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

入力引き数

表 194. verify_my_register_nickname_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_nick_info	Nickname_Info*	CREATE NICKNAME ステートメントからの情報。

出力引き数

表 195. verify_my_register_nickname_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	Nickname_Info**	ラッパーにより追加される情報。

戻り値 戻りコード。0 は成功を示します。

verify_my_alter_nickname_info 関数

目的 ALTER NICKNAME ステートメントからの情報を検証します。

使用法 このメンバー関数は、ラッパーにより、ラッパー固有の nickname サブクラスにインプリメントされる場合があります。ラッパー固有のニックネーム・オプションまたは列オプションがサポートされる場合は、このメンバー関数をインプリメントする必要があります。 verify_my_alter_nickname_info 関数はトラステッド・プロセス・スペースの一部であるため、このメンバー関数はリモート・データ・ソースと対話できません。

ラッパーは、a_delta_info オブジェクトを割り振る前に、すでに割り振られていないか検査します。

構文

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

入力引き数

表 196. verify_my_alter_nickname_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_nick_info	Nickname_Info*	ALTER NICKNAME ステートメントからの情報。

出力引き数

表 197. verify_my_alter_nickname_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	Nickname_Info**	ラッパーにより追加される情報。

戻り値 戻りコード。0 は成功を示します。

get_card 関数

目的 ニックネームのカーディナリティーを戻します。カーディナリティーは DB2 Information Integrator のシステム・カタログに保管されます。

構文

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

入力引き数

なし。

出力引き数

表 198. get_card メンバー関数の出力引き数

名前	データ・タイプ	説明
a_cardinality	sqlint64*	カーディナリティー。

戻り値 なし。

get_advance_cost 関数

目的 デフォルトのコスト・モデルで使用される ADVANCE_COST ニックネーム・オプションの値を戻します。

構文

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

入力引き数

なし。

出力引き数

なし。

戻り値 ADVANCE_COST ニックネーム・オプションの値。ニックネームに対してオプションが指定されていない場合はデフォルト値。

get_setup_cost 関数

目的 デフォルトのコスト・モデルで使用される SETUP_COST ニックネーム・オプションの値を戻します。

構文

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

入力引き数
なし。

出力引き数
なし。

戻り値 SETUP_COST ニックネーム・オプションの値。ニックネームに対してオプションが指定されていない場合はデフォルト値。

get_submission_cost 関数

目的 デフォルトのコスト・モデルで使用される SUBMISSION_COST ニックネーム・オプションの値を戻します。

構文

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

入力引き数
なし。

出力引き数
なし。

戻り値 SUBMISSION_COST ニックネーム・オプションの値。ニックネームに対してオプションが指定されていない場合はデフォルト値。

関連情報:

- 106 ページの『C++ API のニックネーム・クラス』

Fenced_Generic_Nickname クラス

このセクションでは、Fenced_Generic_Nickname クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Fenced_Generic_Nickname クラスは、fenced (非トラステッド) プロセス・スペース内のニックネームを処理します。このクラスは、CREATE NICKNAME ステートメントからの情報を検証します。

Fenced_Generic_Nickname クラスは、C++ API のニックネーム・クラスの 1 つです。

使用法 このクラスは、ラッパーによりサブクラス化され、Fenced_Generic_Server のラッパー固有サブクラスの create_nickname() メソッドにインスタンス化されます。

ファイル

sqlqg_fenced_generic_nickname.h

データ・メンバー

以下の表に、Fenced_Generic_Nickname クラスとともに使用できるデータ・メンバーをリストします。

表 199. Fenced_Generic_Nickname クラスのデータ・メンバー

名前	データ・タイプ	説明
name	sqluint8*	ニックネームの名前 (ローカル) を含むヌル終了ストリング。
schema	sqluint8*	スキーマの名前 (ローカル) を含むヌル終了ストリング。
server	Server*	このニックネームを所有するサーバー・オブジェクトへのポインター。

コンストラクターおよびメンバー関数

以下の表で、Fenced_Generic_Nickname クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 200. Fenced_Generic_Nickname クラスのコンストラクター

コンストラクター	説明
Fenced_Generic_Nickname	Fenced_Generic_Nickname のインスタンスを構成します。

表 201. Fenced_Generic_Nickname クラスのメンバー関数

メンバー関数	説明
get_local_name	ヌル終了したローカル・ニックネーム名へポインターを戻します。
get_local_schema	ヌル終了したローカル・スキーマ名へポインターを戻します。
get_server	収容サーバー・オブジェクトへポインターを戻します。
is_reserved_nickname_option	特定のオプション名が DB2 に組み込まれたニックネーム・オプションの 1 つであるかどうか識別します。
is_reserved_column_option	特定のオプション名が DB2 に組み込まれた列オプションの 1 つであるかどうか識別します。
initialize_my_nickname	カタログからの有効な情報を使用してニックネームを初期化します。
verify_my_register_nickname_info	CREATE NICKNAME ステートメントからの情報を検証します。

Fenced_Generic_Nickname コンストラクター

目的 Fenced_Generic_Nickname のインスタンスを構成します。

構文

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

入力引き数

表 202. Fenced_Generic_Nickname コンストラクターの入力引き数

名前	データ・タイプ	説明
a_schema	sqluint8*	ヌル終了スキーマ名。
a_name	sqluint8*	ヌル終了ニックネーム名 (ローカル)。
a_server	Fenced_Generic_Server*	収容サーバー・オブジェクトへのポインター。

出力引き数

表 203. Fenced_Generic_Nickname コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 なし。

get_local_name 関数

目的 ヌル終了したローカル・ニックネーム名へポインターを戻します。

構文

```
sqluint8* get_local_name ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 ヌル終了ニックネーム名。

get_local_schema 関数

目的 ヌル終了したローカル・スキーマ名へポインターを戻します。

構文

```
sqluint8* get_local_schema ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 ヌル終了スキーマ名。

get_server 関数

目的 収容サーバー・オブジェクトへポインターを戻します。

構文

```
Server* get_server ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 収容サーバー・オブジェクトへのポインター。

is_reserved_nickname_option 関数

目的 特定のオプション名が DB2 に組み込まれたニックネーム・オプションの 1 つであるかどうか識別します。

構文

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

入力引き数

表 204. is_reserved_nickname_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	ヌル終了したオプション名ストリング。

出力引き数
なし。

戻り値 0 は、そのオプションが予約済みオプションではないことを示します。0 以外の場合は、そのオプションが予約済みニックネーム・オプションであることを示します。

is_reserved_column_option 関数

目的 特定のオプション名が DB2 に組み込まれた列オプションの 1 つであるかどうか識別します。

構文

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

入力引き数

表 205. is_reserved_column_option メンバー関数の入力引き数

名前	データ・タイプ	説明
a_opt_name	sqluint8*	ヌル終了したオプション名ストリング。

出力引き数
なし。

戻り値 0 は、そのオプションが予約済みオプションではないことを示します。0 以外の場合は、そのオプションが予約済み列オプションであることを示します。

initialize_my_nickname 関数

目的 カタログからの有効な情報を使用してニックネームを初期化します。

使用法 ラッパーはこのメンバー関数をラッパー固有の fenced nickname サブクラスにインプリメントできます。

構文

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

入力引き数

表 206. initialize_my_nickname メンバー関数の入力引き数

名前	データ・タイプ	説明
a_nick_info	Nickname_Info*	カタログからの情報。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

verify_my_register_nickname_info 関数

目的 CREATE NICKNAME ステートメントからの情報を検証します。

使用法 ラッパーはこのメンバー関数をラッパー固有の fenced nickname サブクラスにインプリメントできます。ラッパー固有のニックネーム・オプションまたは列オプションがサポートされる場合は、このメソッドまたは unfenced クラスの同じメソッドをインプリメントする必要があります。ニックネーム情報の検証時に、ラッパーがリモート・データ・ソースとの対話を必要とする場合は、このメソッドをインプリメントしてください。

ラッパーは、a_delta_info オブジェクトを割り振る前に、すでに割り振られていないか検査します。このメソッドは、
Unfenced_Generic_Nickname::verify_my_register_nickname_info が呼び出される前に呼び出されます。

構文

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

入力引き数

表 207. verify_my_register_nickname_info メンバー関数の入力引き数

名前	データ・タイプ	説明
a_nick_info	Nickname_Info*	CREATE NICKNAME ステートメントからの情報。

出力引き数

表 208. verify_my_register_nickname_info メンバー関数の出力引き数

名前	データ・タイプ	説明
a_delta_info	Nickname_Info**	ラッパーにより追加される情報。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 106 ページの『C++ API のニックネーム・クラス』

Remote_Connection クラス

このセクションでは、Remote_Connection クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Remote_Connection クラスは、DB2 とリモート・データ・ソース間の接続を処理します。このクラスは接続の管理、およびリモート・オペレーション・オブジェクトの作成と保守を行います。

Remote_Connection クラスは C++ API の接続クラスです。

使用法 ラッパーはこのクラスをサブクラス化し、ラッパー固有の接続サブクラスを作成できます。このクラスはラッパー固有の Fenced_Generic_Server サブクラスの create_remote_connection() メソッドによりインスタンス化されます。

ファイル

sqlqg_connection.h

データ・メンバー

以下の表に、Remote_Connection クラスとともに使用できるデータ・メンバーをリストします。

表 209. Remote_Connection クラスのデータ・メンバー

名前	データ・タイプ	説明
kind	connection_kind	接続の種類 (1 フェーズまたは 2 フェーズ)。
server	FencedServer*	この接続を保持するサーバーへのポインター。
user	FencedRemote_User*	この接続に関連するユーザー。
connect	unsigned short	接続がアクティブであることを示すフラグ。

コンストラクターおよびメンバー関数

以下の表で、Remote_Connection クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 210. Remote_Connection クラスのコンストラクター

コンストラクター	説明
Remote_Connection	Remote_Connection オブジェクトを構成します。

表 211. Remote_Connection クラスのメンバー関数

メンバー関数	説明
connect	リモート・データ・ソースとの接続を確立します。
disconnect	リモート・データ・ソースとの接続を破棄します。
is_connected	接続がアクティブか非アクティブか示します。
create_remote_query	Remote_Query の適切なサブクラスを作成します。
create_remote_passthru	Remote_Passthru の適切なサブクラスを作成します。
get_server	収容サーバーにポインターを戻します。
get_user	関連する Remote_User オブジェクトにポインターを戻します。

表 211. Remote_Connection クラスのメンバー関数 (続き)

メンバー関数	説明
connect	トランザクションが正常に完了し、リモート・データ・ソースがトランザクションをコミットすることを示します。
rollback	トランザクションが失敗し、リモート・データ・ソースがトランザクションをロールバックすることを示します。

Remote_Connection コンストラクター

目的 Remote_Connection オブジェクトを構成します。

構文

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

入力引き数

表 212. Remote_Connection コンストラクターの入力引き数

名前	データ・タイプ	説明
FencedServer*	remote_server	この接続を保持するサーバー。
remote_user	FencedRemote_User*	この接続に関連するユーザー。
k	connection_kind	接続の種類。

出力引き数

表 213. Remote_Connection コンストラクターの出力引き数

名前	データ・タイプ	説明
rc	sqlint32*	戻りコードへのポインター。0 は成功を示します。

戻り値 なし。

connect 関数

目的 リモート・データ・ソースとの接続を確立します。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、リモート・データ・ソースとの接続を確立します。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Connection サブクラスにインプリメントされる必要があります。デフォルトでは、エラーをレポートします。

構文

```
virtual sqlint32 connect ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

disconnect 関数

目的 リモート・データ・ソースとの接続を破棄します。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、リモート・データ・ソースとの接続をクローズします。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Connection サブクラスにインプリメントされる必要があります。デフォルトでは、エラーをレポートします。

構文

```
virtual sqlint32 disconnect ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

is_connected 関数

目的 接続がアクティブか非アクティブか示します。

構文

```
unsigned short is_connected ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 接続がアクティブである場合は TRUE。アクティブでない場合は FALSE。

create_remote_query 関数

目的 Remote_Query の適切なサブクラスを作成します。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、ラッパー固有の Remote_Query サブクラスをインスタンス化します。ラッパーはこのメンバー関数をラッパー固有の Remote_Connection サブクラスにインプリメントする必要があります。

構文

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

入力引き数

表 214. create_remote_query メンバー関数の入力引き数

名前	データ・タイプ	説明
runtime_query	Runtime_Operation*	操作を記述する DB2 内部構造。
query	Remote_Query**	割り振られた Remote_Query オブジェクトへのポインター。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

create_remote_passthru 関数

目的 Remote_Passthru の適切なサブクラスを作成します。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、ラッパー固有の Remote_Passthru サブクラスをインスタンス化します。ラッパーはこのメンバー関数をラッパー固有の Remote_Connection サブクラスにインプリメントできます。このメンバー関数は、ラッパーが PASSTHRU モードをサポートする場合のみ必要となります。

構文

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

入力引き数

表 215. create_remote_passthru メンバー関数の入力引き数

名前	データ・タイプ	説明
runtime_passthru	Runtime_Operation*	操作を記述する DB2 内部構造。
query	Runtime_Passthru**	割り振られた Remote_Passthru オブジェクトへのポインター。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_server 関数

目的 収容サーバーにポインターを戻します。

構文

```
FencedServer* get_server()
```

入力引き数

なし。

出力引き数

なし。

戻り値 サーバーへのポインター。

get_user 関数

目的 関連する Remote_User オブジェクトにポインターを戻します。

構文

```
FencedRemote_User* get_user ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 リモート・ユーザー・オブジェクトへのポインター。

commit 関数

目的 トランザクションが正常に完了し、リモート・データ・ソースがトランザクションをコミットすることを示します。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、トランザクションの正常な完了を示します。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Connection サブクラスにインプリメントされる必要があります。デフォルトでは、エラーをレポートします。

構文

```
sqlint32 commit ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

rollback 関数

目的 トランザクションが失敗し、リモート・データ・ソースがトランザクションをロールバックすることを示します。

使用法 このメンバー関数は DB2 Information Integrator に呼び出され、トランザクションが失敗し、リモート・データ・ソースがトランザクションをロールバックすることを示します。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Connection サブクラスにインプリメントされる場合があります。デフォルトでは、エラーをレポートします。

構文

```
sqlint32  rollback ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 78 ページの『C++ API のサーバー・クラス』

C++ API の操作クラス

以下の表で、C++ API の各操作クラスについて説明します。

表 216. 操作クラス

クラス名	説明
Remote_Query	このクラスは、リモート・データ・ソースで照会を実行するのに必要な情報をカプセル化します。ラッパーは、操作をインプリメントするためにこのクラスをサブクラス化する必要があります。
Remote_Passthru	このクラスは、リモート・データ・ソースでのパススルー操作を処理します。ラッパーは、パススルー操作をサポートするために Remote_Passthru クラスをサブクラス化する必要があります。

関連情報:

- 128 ページの『Remote_Query クラス』
- 138 ページの『Remote_Passthru クラス』

Remote_Query クラス

このセクションでは、Remote_Query クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Remote_Query クラスは、リモート・データ・ソースで照会を実行するのに必要な情報をカプセル化します。ラッパーは、操作をインプリメントするためにこのクラスをサブクラス化する必要があります。

Remote_Query クラスは、C++ API の操作クラスの 1 つです。

使用法 このクラスはラッパーにより、ラッパー固有の Remote_Connection サブクラスの create_remote_query() メソッドにインスタンス化されます。

ファイル

sqlqg_operation.h

親クラス

Remote_Operation

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、Remote_Query クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 217. Remote_Query クラスのコンストラクター

コンストラクター	説明
Remote_Query	Remote_Query オブジェクトを構成します。

表 218. Remote_Query クラスのメンバー関数

メンバー関数	説明
get_connection	この操作を所有する接続へのポインターを戻します。
get_server	この操作を所有するサーバーへのポインターを戻します。
get_input_data	操作への入力値のリストへのポインターを戻します。
get_output_data	出力データ・バッファーのリストへのポインターを戻します。
get_exec_desc	この照会の実行記述子のポインターと長さを戻します。
open	照会をオープンします。
reopen	照会を再オープンします。
fetch	単一行を取り出します。
close	照会をクローズします。
report_eof	取り出し中にファイルの終わり状態をレポートします。
get_status	現在の照会状況を戻します。
set_status	現在の照会状況を設定します。
fetch_lob	LOB データを DB2 が提供する特殊な LOB バッファーに戻します。
set_lob_next	非 LOB 列の処理を中断し、1 つ以上の LOB 列の処理を開始します。
lob_data_ready	ラッパーが LOB データのセクションをバッファーに転送したことを示します。

Remote_Query コンストラクター

目的 Remote_Query オブジェクトを構成します。

構文

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

入力引き数

表 219. Remote_Query コンストラクターの入力引き数

名前	データ・タイプ	説明
a_active_connection	Remote_Connection*	この照会に使用する接続。
a_runtime_info	Runtime_Operation*	この照会の内部データ。

出力引き数

表 220. Remote_Query コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコード。0 は成功を示します。

戻り値 なし。

get_connection 関数

定義場所

Remote_Operation

目的 この操作を所有する接続へのポインターを戻します。

構文

```
Remote_Connection* get_connection ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 Remote_Connection オブジェクトへのポインター。

get_server 関数

定義場所

Remote_Operation

目的 この操作を所有するサーバーへのポインターを戻します。

構文

```
Fenced_Generic_Server* get_server ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 サーバー・オブジェクトへのポインター。

get_input_data 関数

定義場所

Remote_Operation

目的 操作への入力値のリストへのポインターを戻します。

構文

```
Runtime_Data_List* get_input_data ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 データ値のリストへのポインター。

get_output_data 関数

定義場所

Remote_Operation

目的 出力データ・バッファーのリストへのポインターを戻します。

構文

```
Runtime_Data_List* get_output_data ()
```

入力引き数

なし。

出力引き数
なし。

戻り値 データ・バッファーのリストへのポインター。

get_exec_desc 関数

定義場所
Remote_Operation

目的 この照会の実行記述子のポインターと長さを戻します。

構文

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

入力引き数
なし。

出力引き数

表 221. get_exec_desc メンバー関数の出力引き数

名前	データ・タイプ	説明
a_exec_desc	void**	実行記述子へのポインター。
a_exec_desc_len	int*	記述子の長さ。

戻り値 なし。

open 関数

目的 照会をオープンします。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、照会を開始します。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Query サブクラスにインプリメントされる必要があります。

構文

```
sqlint32 open ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

reopen 関数

目的 照会を再オープンします。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、新しいパラメーター値を使用して照会を開始します。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Query サブクラスにインプリメントされる必要があります。

構文

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

入力引き数

表 222. reopen メンバー関数の入力引き数

名前	データ・タイプ	説明
a_status	sqlint16	使用されません。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

fetch 関数

目的 単一行を取り出します。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、リモート照会から非 LOB データの行を取り出します。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Query サブクラスにインプリメントされる必要があります。

構文

```
sqlint32 fetch ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

close 関数

目的 照会をクローズします。

使用法 DB2 Information Integrator はこのメンバー関数を呼び出し、照会カーソルをクローズします。このメンバー関数は、ラッパーにより、ラッパー固有の Remote_Query サブクラスにインプリメントされる必要があります。

構文

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

入力引き数

表 223. close メンバー関数の入力引き数

名前	データ・タイプ	説明
a_status	sqlint16	状況 (SQLQG_CLOSE_EOS、SQLQG_CLOSE_EOA、または SQLQG_CLOSE_EOQ)。SQLQG_CLOSE_EOS は、reopen() 関数を呼び出すためにラッパーが現在の状態を保持することを示します。 SQLQG_CLOSE_EOA は、ラッパーがすべての必要な処理を完了できることを示します。SQLQG_CLOSE_EOQ は使用されません。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

report_eof 関数

目的 取り出し中にファイルの終わり状態をレポートします。

使用法 このメンバー関数は、fetch() メソッド中にラッパー固有の Remote_Query クラスにより呼び出され、最終行が取り出されたことを示します。

構文

```
sqlint32 report_eof ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 戻りコード。このコードは、fetch() から戻されます。

get_status 関数

目的 現在の照会状況を戻します。

構文

```
sqluint8 get_status ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 状況 (SQLQG_UNREADY、SQLQG_READY、SQLQG_OPEN、または SQLQG_EOF)。

set_status 関数

目的 現在の照会状況を設定します。

構文

```
void set_status (sqluint8 a_new_status )
```

入力引き数

表 224. set_status メンバー関数の入力引き数

名前	データ・タイプ	説明
a_new_status	sqluint8	状況 (SQLQG_UNREADY、SQLQG_READY、SQLQG_OPEN、または SQLQG_EOF)。

出力引き数
なし。

戻り値 なし。

fetch_lob 関数

目的 LOB データを DB2 が提供する特殊な LOB バッファーに戻します。

構文

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

入力引き数

表 225. fetch_lob メンバー関数の入力引き数

名前	データ・タイプ	説明
a_buffer	unsigned char*	DB2 が提供する LOB データ・バッファーのアドレス。
a_buffer_size	sqluint32	バッファーのサイズ (バイト)。
a_bytes_written	sqluint32	これまでの fetch_lob 関数呼び出しにより現行列に書き込まれたバイト数の合計。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

set_lob_next 関数

目的 非 LOB 列の処理を中断し、1 つ以上の LOB 列の処理を開始します。この関数は、fetch 関数により呼び出されます。

構文

```
void set_lob_next (void)
```

入力引き数

なし。

出力引き数

なし。

戻り値 なし。

lob_data_ready 関数

目的 ラッパーが LOB データのセクションをバッファーに転送したことを示します。

構文

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

入力引き数

表 226. lob_data_ready メンバー関数の入力引き数

名前	データ・タイプ	説明
a_col_number	sqlint32	ラッパーが LOB バッファーに書き込むデータを取得した列。
a_bytes_ready	sqluint32	ラッパーが LOB バッファーにコピーしたバイト数。
a_status	sqlqg_lob_status	状況を示す列挙データ・タイプ。ラッパーが列の最終セクションを処理している場合は LOB_LAST になります。それ以外の場合は LOB_MORE になります。
a_next	sqlqg_lob_intent	ラッパーが次に呼び出すのが fetch 関数と fetch_lob 関数のどちらであるか示す列挙データ・タイプ。有効な値は、LOB_NEXT、NON_LOB_NEXT、および ROW_DONE です。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 128 ページの『C++ API の操作クラス』

Remote_Passthru クラス

このセクションでは、Remote_Passthru クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Remote_Passthru クラスは、リモート・データ・ソースでのパススルー操作を処理します。ラッパーは、パススルー操作をサポートするために Remote_Passthru クラスをサブクラス化する必要があります。

Remote_Passthru クラスは、C++ API の操作クラスの 1 つです。

使用法 このクラスは、ラッパー固有の Remote_Connection サブクラスの create_remote_passthru() メソッドにインスタンス化されます。

ファイル

sqlqg_operation.h

親クラス

Remote_Operation

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、Remote_Passthru クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 227. Remote_Passthru クラスのコンストラクター

コンストラクター	説明
Remote_Passthru	Remote_Passthru オブジェクトを構成します。

表 228. Remote_Passthru クラスのメンバー関数

メンバー関数	説明
get_connection	この操作を所有する接続へのポインターを戻します。
get_server	この操作を所有するサーバーへのポインターを戻します。
get_input_data	操作への入力値のリストへのポインターを戻します。

表 228. Remote_Passthru クラスのメンバー関数 (続き)

メンバー関数	説明
get_output_data	出力データ・バッファーのリストへのポインターを戻します。
get_SQL_statement	パススルーの実行に使用するステートメントを戻します。
report_eof	取り出し中にファイルの終わり状態をレポートします。
prepare	リモートにおけるパススルー操作を準備します。
describe	リモートにおけるパススルー操作を記述します。
execute	リモートにおけるパススルー操作を実行します。
open	リモートにおけるパススルー操作のカーソルをオープンします。
fetch	リモートにおけるパススルー操作で行を取り出します。
close	リモートにおけるパススルー操作のカーソルをクローズします。

Remote_Passthru コンストラクター

目的 Remote_Passthru オブジェクトを構成します。

構文

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

入力引き数

表 229. Remote_Passthru コンストラクターの入力引き数

名前	データ・タイプ	説明
a_active_connection	Remote_Connection*	この照会に使用する接続。
a_runtime_info	Runtime_Operation*	照会の内部データ。

出力引き数

表 230. Remote_Passthru コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコード。0 は成功を示します。

戻り値 なし。

get_connection 関数

定義場所

Remote_Operation

目的 この操作を所有する接続へのポインターを戻します。

構文

```
Remote_Connection* get_connection ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 Remote_Connection オブジェクトへのポインター。

get_server 関数

定義場所

Remote_Operation

目的 この操作を所有するサーバーへのポインターを戻します。

構文

```
Fenced_Generic_Server* get_server ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 サーバー・オブジェクトへのポインター。

get_input_data 関数

定義場所

Remote_Operation

目的 操作への入力値のリストへのポインターを戻します。

構文

```
Runtime_Data_List* get_input_data ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 データ値のリストへのポインター。

get_output_data 関数

定義場所

Remote_Operation

目的 出力データ・バッファーのリストへのポインターを戻します。

構文

```
Runtime_Data_List* get_output_data ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 データ・バッファーのリストへのポインター。

get_SQL_statement 関数

定義場所

Remote_Operation

目的 パススルーの実行に使用するステートメントを戻します。

構文

```
char* get_SQL_statement ()
```

入力引き数

なし。

出力引き数
なし。

戻り値 ステートメント (ヌル終了)。

report_eof 関数

定義場所
Remote_Operation

目的 取り出し中にファイルの終わり状態をレポートします。

構文

```
sqlint32 report_eof ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 戻りコード。このコードは、fetch() から戻されます。

prepare 関数

目的 リモートにおけるパススルー操作を準備します。

構文

```
sqlint32 prepare (Runtime_Data_Desc_List* a_data_description_list)
```

入力引き数
なし。

出力引き数

表 231. prepare メンバー関数の出力引き数

名前	データ・タイプ	説明
a_data_description_list	Runtime_Data_Desc_List*	パススルー操作の結果を示す Runtime_Data_Desc のリスト。

戻り値 戻りコード。0 は成功を示します。

describe 関数

目的 リモートにおけるパススルー操作を記述します。

構文

```
sqlint32 describe (Runtime_Data_Desc_List* a_data_description_list)
```

入力引き数

なし。

出力引き数

表 232. describe メンバー関数の出力引き数

名前	データ・タイプ	説明
a_data_description_list	Runtime_Data_Desc_List*	パススルー操作の結果を示す Runtime_Data_Desc のリスト。

戻り値 戻りコード。0 は成功を示します。

execute 関数

目的 リモートにおけるパススルー操作を実行します。

構文

```
sqlint32 execute ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

open 関数

目的 リモートにおけるパススルー操作のカーソルをオープンします。

構文

```
sqlint32 open ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

fetch 関数

目的 リモートにおけるパススルー操作で行を取り出します。

構文

```
sqlint32 fetch ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

close 関数

目的 リモートにおけるパススルー操作のカーソルをクローズします。

構文

```
sqlint32 close ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 128 ページの『C++ API の操作クラス』

C++ API の要求クラス

以下の表で、C++ API の各要求クラスについて説明します。

表 233. 要求クラス

クラス名	説明
Request	このクラスは、ラッパーにより分析および処理される照会フラグメントをカプセル化します。

表 233. 要求クラス (続き)

クラス名	説明
Reply	このクラスは、ラッパーにより処理される照会の一部を処理します。ラッパーがデフォルト以外のコスト・モデルを使用する場合、このクラスはサブクラス化される場合があります。
Request_Exp	このクラスは SQL 式ノードを記述します。SQL 式ノードは、ヘッド式 (選択リスト) または述部の一部のいずれかです。
Request_Exp_Type	このクラスは、Request_Exp ノードのデータ・タイプ記述子を処理します。
Request_Constant	このクラスは、Request_Exp::constant タイプを持つ Request_Exp ノードのデータ値を処理します。
Predicate_List	このクラスは述部のリストをカプセル化し、RRC プロトコルにより使用されます。

関連情報:

- 168 ページの『Request_Exp クラス』
- 145 ページの『Request クラス』
- 177 ページの『Request_Constant クラス』
- 174 ページの『Request_Exp_Type クラス』
- 152 ページの『Reply クラス』
- 181 ページの『Predicate_List クラス』

Request クラス

このセクションでは、Request クラスについて説明し、その各メンバー関数について解説します。

概要

Request クラスは、ラッパーにより分析および処理される照会フラグメントをカプセル化します。

Request クラスは、C++ API の要求クラスの 1 つです。

使用法 このクラスは、ラッパーによりインスタンス化されません。

ファイル

sqlqg_request.h

親クラス

Parsed_Query_Fragment

データ・メンバー

なし。

メンバー関数

以下の表で、Request クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 234. Request クラスのメンバー関数

メンバー関数	説明
get_number_of_quantifiers	この照会の数量詞の数を戻します。
get_number_of_predicates	この照会の述部の数を戻します。
get_number_of_head_exp	この照会のヘッド (選択リスト) 式の数を戻します。
get_quantifier_handle	この照会の *n* 番目の数量詞のハンドルを戻します。
get_predicate_handle	この照会の *n* 番目の述部のハンドルを戻します。
get_head_exp_handle	この照会の *n* 番目のヘッド式のハンドルを戻します。
get_nickname	特定のハンドルに関連するニックネーム・オブジェクトを戻します。
get_head_exp	特定のハンドルに関連するヘッド式を戻します。
get_predicate	特定のハンドルに関連する述部式を戻します。
get_distinct	照会に対する DISTINCT フラグを戻します。

get_number_of_quantifiers 関数

定義場所

Parsed_Query_Fragment

目的 この照会の数量詞 (ニックネームや表式) の数を戻します。

構文

```
int get_number_of_quantifiers ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数量詞の数。

get_number_of_predicates 関数

定義場所
Parsed_Query_Fragment

目的 この照会の述部の数を戻します。

構文

```
int get_number_of_predicates ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 述部の数。

get_number_of_head_exp 関数

定義場所
Parsed_Query_Fragment

目的 この照会のヘッド (選択リスト) 式の数を戻します。

構文

```
int get_number_of_head_exp ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 ヘッド式の数。

get_quantifier_handle 関数

定義場所
Parsed_Query_Fragment

目的 この照会の *n* 番目の数量詞のハンドルを戻します。

構文

```
sqlint32 get_quantifier_handle (int  a_quant_pos,
                                int* a_quant_handle)
```

入力引き数

表 235. get_quantifier_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_quant_pos	int	リスト内のハンドルの位置 (1 から開始)。

出力引き数

表 236. get_quantifier_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_quant_handle	int*	ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_predicate_handle 関数

定義場所

Parsed_Query_Fragment

目的 この照会の *n* 番目の述部のハンドルを戻します。

構文

```
sqlint32 get_predicate_handle (int  a_pred_pos,
                               int* a_pred_handle )
```

入力引き数

表 237. get_predicate_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_pos	int	リスト内のハンドルの位置 (1 から開始)。

出力引き数

表 238. get_predicate_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_handle	int*	ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_head_exp_handle 関数

定義場所

Parsed_Query_Fragment

目的 この照会の *n* 番目のヘッド式のハンドルを戻します。

構文

```
sqlint32 get_head_exp_handle (int  a_head_exp_pos,
                              int* a_head_exp_handle)
```

入力引き数

表 239. get_head_exp_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_head_exp_pos	int	リスト内のハンドルの位置 (1 から開始)。

出力引き数

表 240. get_head_exp_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_head_exp_handle	int*	ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_nickname 関数

定義場所

Parsed_Query_Fragment

目的 特定のハンドルに関連するニックネーム・オブジェクトを戻します。

構文

```
sqlint32 get_nickname (int                          a_quant_handle,
                       Unfenced_Generic_Nickname** a_nickname)
```

入力引き数

表 241. get_nickname メンバー関数の入力引き数

名前	データ・タイプ	説明
a_quant_handle	int	ニックネームのハンドル。

出力引き数

表 242. get_nickname メンバー関数の出力引き数

名前	データ・タイプ	説明
a_nickname	Unfenced_Generic_Nickname**	ニックネーム・オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_head_exp 関数

定義場所

Parsed_Query_Fragment

目的 特定のハンドルに関連するヘッド式を戻します。

構文

```
sqlint32 get_head_exp (int           a_head_exp_handle,
                       Request_Exp** a_head_exp)
```

入力引き数

表 243. get_head_exp メンバー関数の入力引き数

名前	データ・タイプ	説明
a_head_exp_handle	int	式のハンドル。

出力引き数

表 244. get_head_exp メンバー関数の出力引き数

名前	データ・タイプ	説明
a_head_exp	Request_Exp**	式オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_predicate 関数

定義場所

Parsed_Query_Fragment

目的 特定のハンドルに関連する述部式を戻します。

構文

```
sqlint32 get_predicate (int           a_pred_handle,
                        Request_Exp** a_pred_exp)
```

入力引き数

表 245. get_predicate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_handle	int	式のハンドル。

出力引き数

表 246. get_predicate メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_exp	Request_Exp**	式オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_distinct 関数

定義場所

Parsed_Query_Fragment

目的 照会に対する DISTINCT フラグを戻します。

構文

```
int get_distinct ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 Distinct フラグ。

関連情報:

- 144 ページの『C++ API の要求クラス』

Reply クラス

このセクションでは、Reply クラスについて説明し、そのコンストラクターおよびメンバー関数について解説します。

概要

Reply クラスは、ラッパーが処理する照会の一部を処理します。ラッパーがデフォルト以外のコスト・モデルを使用する場合、このクラスはサブクラス化される場合があります。

Reply クラスは、C++ API の要求クラスの 1 つです。

使用法 このクラスは、Unfenced_Generic_Server クラスの create_reply() メソッドによりインスタンス化されます。ラッパーが Reply クラスのサブクラスをインプリメントする場合、create_reply() メソッドが Unfenced_Generic_Server のラッパー固有サブクラスにオーバーライドされます。

ファイル

sqlqg_request.h

親クラス

Parsed_Query_Fragment

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、Reply クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 247. Reply クラスのコンストラクター

コンストラクター	説明
Reply	空の応答オブジェクトを構成します。

表 248. Reply クラスのメンバー関数

メンバー関数	説明
get_number_of_quantifiers	この照会の数量詞の数を戻します。
get_number_of_predicates	この照会の述部の数を戻します。
get_number_of_head_exp	この照会のヘッド (選択リスト) 式の数を戻します。
get_quantifier_handle	この照会の *n* 番目の数量詞のハンドルを戻します。

表 248. *Reply* クラスのメンバー関数 (続き)

メンバー関数	説明
get_predicate_handle	この照会の *n* 番目の述部のハンドルを戻します。
get_head_exp_handle	この照会の *n* 番目のヘッド式のハンドルを戻します。
get_nickname	特定のハンドルに関連するニックネーム・オブジェクトを戻します。
get_head_exp	特定のハンドルに関連するヘッド式を戻します。
get_predicate	特定のハンドルに関連する述部式を戻します。
get_distinct	照会に対する DISTINCT フラグを戻します。
set_distinct	照会に対する DISTINCT フラグを設定します。
add_head_exp	応答にヘッド式を追加します。
add_predicate	応答に述部式を追加します。
add_quantifier	応答に数量詞を追加します。
add_order_entry	応答に ORDER BY 指定を追加します。
get_number_of_order_entries	応答の ORDER BY エントリーの数を戻します。
get_order_entry	特定の ORDER BY エントリーを戻します。
get_exec_desc	応答に関連する実行記述子を戻します。
set_exec_desc	応答に実行記述子を保管します。
get_parameter_order	パラメーター・ハンドルのリストを戻します。
Cardinality	照会フラグメントのカーディナリティーを戻します。
total_cost	照会フラグメントの合計コストを戻します。
re_exec_cost	照会フラグメントの再実行のコスト。
first_tuple_cost	最初のタプルを取り出すコストを戻します。
all_costs	コスト計算値をすべて戻します。

表 248. Reply クラスのメンバー関数 (続き)

メンバー関数	説明
next	応答のチェーンの次の応答へのポインターを戻します。
set_next_reply	新しい応答を現行応答にリンクさせます。
get_server	応答を所有するサーバーへのポインターを戻します。

Reply コンストラクター

目的 空の応答オブジェクトを構成します。

構文

```
Reply (Request*                 a_rq,
       Unfenced_Generic_Server* a_server,
       sqlint32*                a_rc)
```

入力引き数

表 249. Reply コンストラクターの入力引き数

名前	データ・タイプ	説明
a_rq	Request*	この応答が派生した要求。
a_server	Unfenced_Generic_Server*	要求の対象サーバーまたは応答プロトコル。

出力引き数

表 250. Reply コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコード。0 は成功を示します。

戻り値 なし。

get_number_of_quantifiers 関数

定義場所

Parsed_Query_Fragment

目的 この照会の数量詞 (ニックネームや表式) の数を戻します。

構文

```
int get_number_of_quantifiers ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数量詞の数。

get_number_of_predicates 関数

定義場所
Parsed_Query_Fragment

目的 この照会の述部の数を戻します。

構文

```
int get_number_of_predicates ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 述部の数。

get_number_of_head_exp 関数

定義場所
Parsed_Query_Fragment

目的 この照会のヘッド (選択リスト) 式の数を戻します。

構文

```
int get_number_of_head_exp ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 ヘッド式の数。

get_quantifier_handle 関数

定義場所
Parsed_Query_Fragment

目的 この照会の *n* 番目の数量詞のハンドルを戻します。

構文

```
sqlint32 get_quantifier_handle (int  a_quant_pos,
                                int* a_quant_handle)
```

入力引き数

表 251. get_quantifier_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_quant_pos	int	リスト内のハンドルの位置 (1 から開始)。

出力引き数

表 252. get_quantifier_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_quant_handle	int*	ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_predicate_handle 関数

定義場所

Parsed_Query_Fragment

目的 この照会の *n* 番目の述部のハンドルを戻します。

構文

```
sqlint32 get_predicate_handle (int  a_pred_pos,
                               int* a_pred_handle)
```

入力引き数

表 253. get_predicate_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_pos	int	リスト内のハンドルの位置 (1 から開始)。

出力引き数

表 254. get_predicate_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_handle	int*	ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_head_exp_handle 関数

定義場所

Parsed_Query_Fragment

目的 この照会の *n* 番目のヘッド式のハンドルを戻します。

構文

```
sqlint32 get_head_exp_handle (int  a_head_exp_pos,
                              int* a_head_exp_handle)
```

入力引き数

表 255. get_head_exp_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_head_exp_pos	int	リスト内のハンドルの位置 (1 から開始)。

出力引き数

表 256. get_head_exp_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_head_exp_handle	int*	ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_nickname 関数

定義場所

Parsed_Query_Fragment

目的 特定のハンドルに関連するニックネーム・オブジェクトを戻します。

構文

```
sqlint32 get_nickname (int                          a_quant_handle,
                       Unfenced_Generic_Nickname** a_nickname)
```

入力引き数

表 257. get_nickname メンバー関数の入力引き数

名前	データ・タイプ	説明
a_quant_handle	int	ニックネームのハンドル。

出力引き数

表 258. get_nickname メンバー関数の出力引き数

名前	データ・タイプ	説明
a_nickname	Unfenced_Generic_Nickname**	ニックネーム・オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_head_exp 関数

定義場所

Parsed_Query_Fragment

目的 特定のハンドルに関連するヘッド式を戻します。

構文

```
sqlint32 get_head_exp (int a          a_head_exp_handle,
                       Request_Exp** a_head_exp)
```

入力引き数

表 259. get_head_exp メンバー関数の入力引き数

名前	データ・タイプ	説明
a_head_exp_handle	int	式のハンドル。

出力引き数

表 260. get_head_exp メンバー関数の出力引き数

名前	データ・タイプ	説明
a_head_exp	Request_Exp**	式オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_predicate 関数

定義場所

Parsed_Query_Fragment

目的 特定のハンドルに関連する述部式を戻します。

構文

```
sqlint32 get_predicate (int           a_pred_handle,
                        Request_Exp** a_pred_exp)
```

入力引き数

表 261. get_predicate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_handle	int	式のハンドル。

出力引き数

表 262. get_predicate メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_exp	Request_Exp**	式オブジェクトへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_distinct 関数

定義場所

Parsed_Query_Fragment

目的 照会に対する DISTINCT フラグを戻します。

構文

```
int get_distinct ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 Distinct フラグ。

set_distinct 関数

定義場所

Parsed_Query_Fragment

目的 照会に対する DISTINCT フラグを設定します。

構文

```
void set_distinct (int a_distinct)
```

入力引き数

表 263. set_distinct メンバー関数の入力引き数

名前	データ・タイプ	説明
a_distinct	int	Distinct フラグ (DISTINCT の場合 1、DISTINCT 以外の場合 0)。

出力引き数

なし。

戻り値 なし。

add_head_exp 関数

目的 応答にヘッド式を追加します。

構文

```
sqlint32 add_head_exp (int a_head_exp_handle)
```

入力引き数

表 264. add_head_exp メンバー関数の入力引き数

名前	データ・タイプ	説明
a_head_exp_handle	int	式のハンドル。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

add_predicate 関数

目的 応答に述部式を追加します。

構文

```
sqlint32 add_predicate_exp (int a_pred_handle)
```

入力引き数

表 265. add_predicate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_handle	int	式のハンドル。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

add_quantifier 関数

目的 応答に数量詞を追加します。

構文

```
sqlint32 add_quantifier (int a_quant_handle)
```

入力引き数

表 266. add_quantifier メンバー関数の入力引き数

名前	データ・タイプ	説明
a_quant_handle	int	数量詞のハンドル。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

add_order_entry 関数

目的 応答に ORDER BY 指定を追加します。

構文

```
sqlint32 add_order_entry (int                    a_gindex,
                          Reply::order_direction a_direction)
```

入力引き数

表 267. add_order_entry メンバー関数の入力引き数

名前	データ・タイプ	説明
a_gindex	int	配列のためのヘッド式索引 (ハンドルではない)。
a_direction	Reply_order_direction	配列の方向 (昇順または降順)。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_number_of_order_entries 関数

目的 応答の ORDER BY エントリーの数を戻します。

構文

```
int get_number_of_order_entries ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 ORDER BY エントリーの数。

get_order_entry 関数

目的 特定の ORDER BY エントリーを戻します。

構文

```
get_order_entry (int                     a_pos,
                 int*                    a_gindexP,
                 Reply::order_direction* a_direction)
```

入力引き数

表 268. get_order_entry メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pos	int	順序エントリーの位置 (1 から開始)。

出力引き数

表 269. get_order_entry メンバー関数の出力引き数

名前	データ・タイプ	説明
a_gindexP	int*	ORDER BY 式のヘッド式索引。
a_direction	Reply::order_direction*	配列の方向。

戻り値 戻りコード。0 は成功を示します。

get_exec_desc 関数

目的 応答に関連する実行記述子を戻します。

構文

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

入力引き数

なし。

出力引き数

表 270. get_exec_desc メンバー関数の出力引き数

名前	データ・タイプ	説明
a_exec_desc	void**	実行記述子へのポインター。
a_exec_desc_size	int*	実行記述子のサイズ。

戻り値 なし。

set_exec_desc 関数

目的 応答に実行記述子を保管します。

使用法 実行記述子のストレージは、Wrapper_Utillities::allocate を使用して割り当てます。

構文

```
void set_exec_desc (void** a_exec_desc,
                    int*   a_exec_desc_size)
```

入力引き数

表 271. set_exec_desc メンバー関数の入力引き数

名前	データ・タイプ	説明
a_exec_desc	void**	実行記述子へのポインター。
a_exec_desc_size	int*	実行記述子のサイズ。

出力引き数

なし。

戻り値 なし。

get_parameter_order 関数

目的 パラメーター・ハンドルのリストを戻します。リストの順序は、Remote_Operation オブジェクト内のパラメーター値の順序に対応します。

構文

```
sqlint32 get_parameter_order (int*  a_number_of_params,
                              int** a_param_handle_array)
```

入力引き数

なし。

出力引き数

表 272. get_parameter_order メンバー関数の出力引き数

名前	データ・タイプ	説明
a_number_of_params	int*	パラメーター・ハンドルの数。
a_param_handle_array	int**	パラメーター・ハンドルの配列。

戻り値 戻りコード。0 は成功を示します。

cardinality 関数

目的 照会フラグメントのカーディナリティーを戻します。

構文

```
sqlint32 cardinality (float* a_cardinality)
```

入力引き数

なし。

出力引き数

表 273. cardinality メンバー関数の出力引き数

名前	データ・タイプ	説明
a_cardinality	float*	カーディナリティー。

戻り値 戻りコード。0 は成功を示します。

total_cost 関数

目的 照会フラグメントの合計コストを戻します。

構文

```
sqlint32 total_cost (float* a_total_cost)
```

入力引き数

なし。

出力引き数

表 274. total_cost メンバー関数の出力引き数

名前	データ・タイプ	説明
a_total_cost	float*	コスト。

戻り値 戻りコード。0 は成功を示します。

re_exec_cost 関数

目的 照会フラグメントの再実行のコスト。

構文

```
sqlint32 re_exec_cost (float* a_re_exec_cost)
```

入力引き数

なし。

出力引き数

表 275. re_exec_cost メンバー関数の出力引き数

名前	データ・タイプ	説明
a_re_exec_cost	float*	コスト。

戻り値 戻りコード。0 は成功を示します。

first_tuple_cost 関数

目的 最初のタプルを取り出すコストを戻します。

構文

```
sqlint32 first_tuple_cost (float* a_first_tuple_cost)
```

入力引き数

なし。

出力引き数

表 276. first_tuple_cost メンバー関数の出力引き数

名前	データ・タイプ	説明
a_first_tuple_cost	float*	コスト。

戻り値 戻りコード。0 は成功を示します。

all_costs 関数

目的 コスト計算値をすべて戻します。

構文

```
sqlint32 all_costs (float* a_total_cost,
                    float* a_first_tuple_cost,
                    float* a_re_exec_cost)
```

入力引き数

なし。

出力引き数

表 277. all_costs メンバー関数の出力引き数

名前	データ・タイプ	説明
a_total_cost	float*	合計コスト。
a_first_tuple_cost	float*	最初のタプルのコスト。
a_re_exec_cost	float*	再実行コスト。

戻り値 戻りコード。0 は成功を示します。

next 関数

目的 応答のチェーンの次の応答へのポインターを戻します。

構文

```
Reply* next ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 次の応答へのポインター。チェーンの末尾の場合は NULL。

set_next_reply 関数

目的 新しい応答を現行応答にリンクさせます。

使用法 ラッパーが、応答をチェーンの末尾に追加することを確認します。これが実行されない場合、メモリー・リークが発生する可能性があります。

構文

```
void set_next_reply (Reply* a_next_reply)
```

入力引き数

表 278. set_next_reply メンバー関数の入力引き数

名前	データ・タイプ	説明
a_next_reply	Reply*	チェーンに追加される応答。

出力引き数
なし。

戻り値 なし。

get_server 関数

目的 この応答を所有するサーバーへのポインターを戻します。

構文

```
Unfenced_Generic_server*  get_server ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 サーバー・オブジェクトへのポインター。

関連情報:

• 144 ページの『C++ API の要求クラス』

Request_Exp クラス

このセクションでは、Request_Exp クラスについて説明し、その各メンバー関数について解説します。

概要

Request_Exp クラスは、式ツリー内のノードを扱います。ノードは列参照、定数値、ホスト・パラメーター、またはオペレーターのいずれかです。

Request_Exp クラスは、C++ API の要求クラスの 1 つです。

使用法 このクラスは、ラッパーによりインスタンス化されません。

ファイル
sqlqg_request.h

データ・メンバー
なし。

タイプ Request_Exp::kind

タイプ enum

値 badkind、column、unbound、constant、oper

メンバー関数

以下の表で、Request_Exp クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 279. Request_Exp クラスのメンバー関数

メンバー関数	説明
get_kind	式ノードの種類を戻します。
get_data_type	式と関連するデータ・タイプを戻します。
get_parent	現行ノードの親式ノードへのポインターを戻します。

表 279. Request_Exp クラスのメンバー関数 (続き)

メンバー関数	説明
get_next_child	同じ親の次の子へのポインターを戻します。
get_handle	式ノードのハンドルを戻します。
get_column_name	列式ノードの列名を戻します。
get_column_quantifier_handle	列数量詞へのハンドルを戻します。
get_value	定数式ノードの値を戻します。
get_first_child	演算子式ノードの最初の子を戻します。
get_number_of_children	演算子式ノードの子の数を戻します。
get_token	演算子式ノードのトークンを戻します。
get_signature	演算子式ノードのフルシグニチャーを戻します。

get_kind 関数

目的 式ノードの種類を戻します。

使用法 このメンバー関数には、sqlint32 get_kind() という構文もあります。この構文は、Request または Reply が使用できない場合に Unfenced_Server::get_selectivity() 式で使用されます。この get_kind() 構文は、列参照とアンバインド参照を区別しません。

構文

```
sqlint32 get_kind (Parsed_Query_Fragment* a_query_fragment,
                   kind*                  a_kind )
```

入力引き数

表 280. get_kind メンバー関数の入力引き数

名前	データ・タイプ	説明
a_query_fragment	Parsed_Query_Fragment*	この式が属する Request または Reply。

出力引き数

表 281. get_kind メンバー関数の出力引き数

名前	データ・タイプ	説明
a_kind	kind*	式ノードの種類。

戻り値 戻りコード。0 は成功を示します。

get_data_type 関数

目的 式と関連するデータ・タイプを戻します。

構文

```
sqlint32 get_data_type (Request_Exp_Type** a_new_type)
```

入力引き数

なし。

出力引き数

表 282. get_data_type メンバー関数の出力引き数

名前	データ・タイプ	説明
a_new_type	Request_Exp_Type**	記述子のタイプへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_parent 関数

目的 現行ノードの親式ノードへのポインターを戻します。

構文

```
Request_Exp* get_parent ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 親ノードへのポインター。最上位ノードの場合は NULL。

get_next_child 関数

目的 同じ親の次の子 (現行ノードの次の兄弟) へのポインターを戻します。

構文

```
Request_Exp* get_next_child ()
```

入力引き数

なし。

出力引き数
なし。

戻り値 次の子へのポインター。他に兄弟がない場合は NULL。

get_handle 関数

目的 式ノードのハンドルを戻します。

構文

```
sqlint32 get_handle (int* a_handle)
```

入力引き数
なし。

出力引き数

表 283. get_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_handle	int*	ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_column_name 関数

目的 列式ノードの列名を戻します。

使用法 このメソッドは、式の種類が Request_Exp::column の場合のみ有効です。

構文

```
sqlint32 get_column_name (char** a_col_name,
                          int*   a_name_length)
```

入力引き数
なし。

出力引き数

表 284. get_column_name メンバー関数の出力引き数

名前	データ・タイプ	説明
a_col_name	char**	列名 (非ヌル終了) へのポインター。
a_name_length	int*	列名の長さ。

戻り値 戻りコード。0 は成功を示します。

get_column_quantifier_handle 関数

目的 列数量詞へのハンドル (表または表式) を戻します。

使用法 このメソッドは、式の種類が Request_Exp::column の場合のみ有効です。

構文

```
sqlint32 get_column_quantifier_handle (int* a_quant_handle)
```

入力引き数

なし。

出力引き数

表 285. get_column_quantifier_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_quant_handle	int*	数量詞ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_value 関数

目的 定数式ノードの値を戻します。

使用法 このメソッドは、式の種類が Request_Exp::constant の場合のみ有効です。

構文

```
sqlint32 get_value (Request_Constant** a_constant_value)
```

入力引き数

なし。

出力引き数

表 286. get_value メンバー関数の出力引き数

名前	データ・タイプ	説明
a_constant_value	Request_Constant**	定数値へのポインター。

戻り値 戻りコード。0 は成功を示します。

get_first_child 関数

目的 演算子式ノードの最初の子を戻します。

使用法 このメソッドは、式の種類が Request_Exp::oper の場合のみ有効です。

構文

```
Request_Exp* get_first_child ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 最初の子へのポインター。子がない場合は NULL。

get_number_of_children 関数

目的 演算子式ノードの子の数を戻します。

使用法 このメソッドは、式の種類が Request_Exp::oper の場合のみ有効です。

構文

```
int get_number_of_children ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 子の数。

get_token 関数

目的 演算子式ノードのトークン (関数名) を戻します。

使用法 このメソッドは、式の種類が Request_Exp::oper の場合のみ有効です。

構文

```
sqlint32 get_token (char** a_operator_token,
                    int*   a_token_length )
```

入力引き数

なし。

出力引き数

表 287. get_token メンバー関数の出力引き数

名前	データ・タイプ	説明
a_operator_token	char**	トークン (非ヌル終了) へのポインター。
a_token_length	int*	トークンの長さ。

戻り値 戻りコード。0 は成功を示します。

get_signature 関数

目的 演算子式ノードのフルシグニチャーを戻します。

使用法 このメソッドは、式の種類が Request_Exp::oper の場合のみ有効です。

構文

```
sqlint32 get_signature (char** a_signature)
```

入力引き数

なし。

出力引き数

表 288. get_signature メンバー関数の出力引き数

名前	データ・タイプ	説明
a_signature	char**	ヌル終了シグニチャーへのポインター。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 144 ページの『C++ API の要求クラス』

Request_Exp_Type クラス

このセクションでは、Request_Exp_Type クラスについて説明し、その各メンバー関数について解説します。

概要

Request_Exp_Type クラスは、Request_Exp ノードのデータ・タイプ記述子を処理します。

Request_Exp_Type クラスは、C++ API の要求クラスの 1 つです。

使用法 このクラスは、ラッパーによりインスタンス化されません。

ファイル

sqlqg_runtime_data_operation.h

データ・メンバー

なし。

メンバー関数

以下の表で、Request_Exp_Type クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 289. Request_Exp_Type クラスのメンバー関数

メンバー関数	説明
get_for_bit_data	FOR BIT DATA フラグを戻します。
get_null_indicator	NULL 可能標識を戻します。
get_data_type	データ・タイプを戻します。
get_maximum_length	タイプの最大長を戻します。
get_precision	数値タイプの精度を戻します。
get_scale	数値タイプの尺度を戻します。
get_codepage	文字タイプのコード・ページを戻します。

get_for_bit_data 関数

目的 データの FOR BIT DATA フラグを戻します。

構文

```
unsigned char get_for_bit_data ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 FOR BIT DATA フラグ (Y または N)。

get_null_indicator 関数

目的 NULL 可能標識を戻します。

構文

```
short get_null_indicator ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 NULL 標識フラグ。SQL_NULLABLE または SQL_NO_NULLS。

get_data_type 関数

目的 データ・タイプを戻します。

構文

```
short get_data_type ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 データ・タイプ (SQL_TYP_xxx)。

get_maximum_length 関数

目的 タイプの最大長を戻します。

構文

```
int get_maximum_length ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 タイプの最大長。

get_precision 関数

目的 数値タイプの精度を戻します。

構文

```
unsigned char get_precision ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数値精度。

get_scale 関数

目的 数値タイプの尺度を戻します。

構文

```
unsigned char get_scale ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数値尺度。

get_codepage 関数

目的 文字タイプのコード・ページを戻します。

構文

```
unsigned short get_codepage ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 コード・ページ。

関連情報:

- 144 ページの『C++ API の要求クラス』

Request_Constant クラス

このセクションでは、Request_Constant クラスについて説明し、その各メンバー関数について解説します。

概要

Request_Constant クラスは、Request_Exp::constant タイプを持つ Request_Exp ノードのデータ値を処理します。

Request_Constant クラスは、C++ API の要求クラスの 1 つです。

使用法 このクラスは、ラッパーによりインスタンス化されません。

ファイル
sqlqg_runtime_data_operation.h

データ・メンバー
なし。

メンバー関数

以下の表で、Request_Constant クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 290. Request_Constant クラスのメンバー関数

メンバー関数	説明
get_actual_length	データ値の実際の長さを戻します。
get_data	データ値へのポインターを戻します。
is_data_null	データ値が NULL かどうか示します。
get_for_bit_data	FOR BIT DATA フラグを戻します。
get_null_indicator	NULL 可能標識を戻します。
get_data_type	データ・タイプを戻します。
get_maximum_length	タイプの最大長を戻します。
get_precision	数値タイプの精度を戻します。
get_scale	数値タイプの尺度を戻します。
get_codepage	文字タイプのコード・ページを戻します。

get_actual_length 関数

目的 データ値の実際の長さを戻します。

構文

```
int get_actual_length ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 実際のデータ長。

get_data 関数

目的 データ値へのポインターを戻します。

構文

```
unsigned char* get_data ()
```

入力引き数

なし。

出力引き数
なし。

戻り値 データ値へのポインター。

is_data_null 関数

目的 データ値が NULL かどうか示します。

構文

```
sqlint32 is_data_null ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 NULL 標識 (TRUE または FALSE)。

get_for_bit_data 関数

目的 データの FOR BIT DATA フラグを戻します。

構文

```
unsigned char get_for_bit_data ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 FOR BIT DATA フラグ (Y または N)。

get_null_indicator 関数

目的 NULL 可能標識を戻します。

構文

```
short get_null_indicator ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 NULL 標識フラグ。SQL_NULLABLE または SQL_NO_NULLS。

get_data_type 関数

目的 データ・タイプを戻します。

構文

```
short get_data_type ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 データ・タイプ (SQL_TYP_xxx)。SQL_TYP_xxx 記号については、sql.h ヘッダー・ファイルを参照。

get_maximum_length 関数

目的 タイプの最大長を戻します。

構文

```
int get_maximum_length ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 タイプの最大長。

get_precision 関数

目的 数値タイプの精度を戻します。

構文

```
unsigned char get_precision ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数値精度。

get_scale 関数

目的 数値タイプの尺度を戻します。

構文

```
unsigned char get_scale ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数値尺度。

get_codepage 関数

目的 文字タイプのコード・ページを戻します。

構文

```
unsigned short get_codepage ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 コード・ページ。

関連情報:

- 144 ページの『C++ API の要求クラス』

Predicate_List クラス

このセクションでは、Predicate_List クラスについて説明し、その各メンバー関数について解説します。

概要

Predicate_List クラスは述部のリストをカプセル化し、RRC プロトコルにより使用されます。

Predicate_List クラスは、C++ API の要求クラスの 1 つです。

使用法 カスタム・コスト・モデルが特定の述部リストに対して get_selectivity() メソッドを呼び出さない限り、ラッパーはこのクラスをインスタンス化しません。

ファイル

sqlqg_request.h

データ・メンバー

なし。

メンバー関数

以下の表で、Predicate_List クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 291. Predicate_List クラスのメンバー関数

メンバー関数	説明
create_empty_predicate_list	空の述部リストをインスタンス化します。
create_and_copy_predicate_list	特定の応答の述部をすべて含む述部リストを作成します。
get_number_of_predicates	リスト内の述部の数を戻します。
get_predicate_handle	リスト内の特定の位置に対する述部ハンドルを戻します。
get_predicate	式ハンドルの述部式を戻します。
get_number_of_applied_predicates	適用済みの述部の数を戻します。
get_applied_predicate_handle	リスト内の特定の位置に対して適用済みの述部ハンドルを戻します。
get_applied_predicate	式ハンドルの適用済み述部式を戻します。
add_predicate	リストに述部を追加します。
add_applied_predicate	適用済み述部リストに述部式を追加します。

create_empty_predicate_list 関数

目的 空の述部リストをインスタンス化します。

構文

```
static sqlint32 create_empty_predicate_list
   (Reply*           a_reply,
    Predicate_List** a_pred_list )
```

入力引き数

表 292. create_empty_predicate_list メンバー関数の入力引き数

名前	データ・タイプ	説明
a_reply	Reply*	述部を保有する応答。

出力引き数

表 293. create_empty_predicate_list メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_list	Predicate_List**	新しい述部リストへのポインター。

戻り値 戻りコード。0 は成功を示します。

create_and_copy_predicate_list 関数

目的 特定の応答の述部をすべて含む述部リストを作成します。

構文

```
static sqlint32 create_and_copy_predicate_list
   (Reply*           a_reply,
    Predicate_List** a_pred_list)
```

入力引き数

表 294. create_and_copy_predicate_list メンバー関数の入力引き数

名前	データ・タイプ	説明
a_reply	Reply*	述部を保有する応答。

出力引き数

表 295. create_and_copy_predicate_list メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_list	Predicate_List**	新しい述部リストへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_number_of_predicates 関数

目的 リスト内の述部の数を戻します。

構文

```
int get_number_of_predicates ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 述部の数。

get_predicate_handle 関数

目的 リスト内の特定の位置に対する述部ハンドルを戻します。

構文

```
sqlint32 get_predicate_handle (int  a_pred_pos,
                               int* a_pred_handle)
```

入力引き数

表 296. get_predicate_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_pos	int	リスト内の述部の位置 (1 から開始)。

出力引き数

表 297. get_predicate_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_handle	int*	述部式ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_predicate 関数

目的 式ハンドルの述部式を戻します。

構文

```
sqlint32 get_predicate (int           a_pred_handle,
                        Request_Exp** a_pred_exp)
```

入力引き数

表 298. get_predicate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_handle	int	述部式ハンドル。

出力引き数

表 299. get_predicate メンバー関数の出力引き数

名前	データ・タイプ	説明
a_pred_exp	Request_Exp**	式ノードへのポインター。

戻り値 戻りコード。0 は成功を示します。

get_number_of_applied_predicates 関数

目的 適用済みの述部の数を戻します。

構文

```
int get_number_of_applied_predicates ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 適用済み述部の数。

get_applied_predicate_handle 関数

目的 リスト内の特定の位置に対して適用済みの述部ハンドルを戻します。

構文

```
sqlint32 get_applied_predicate_handle (int  a_applied_pred_pos,
                                       int* a_applied_pred_handle)
```

入力引き数

表 300. get_applied_predicate_handle メンバー関数の入力引き数

名前	データ・タイプ	説明
a_applied_pred_pos	int	リスト内の述部の位置 (1 から開始)。

出力引き数

表 301. get_applied_predicate_handle メンバー関数の出力引き数

名前	データ・タイプ	説明
a_applied_pred_handle	int*	適用済み述部式ハンドル。

戻り値 戻りコード。0 は成功を示します。

get_applied_predicate 関数

目的 式ハンドルの適用済み述部式を戻します。

構文

```
sqlint32 get_applied_predicate (int           a_applied_pred_handle,
                                Request_Exp** a_applied_pred_exp)
```

入力引き数

表 302. get_applied_predicate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_applied_pred_handle	int	適用済み述部式ハンドル。

出力引き数

表 303. get_applied_predicate メンバー関数の出力引き数

名前	データ・タイプ	説明
a_applied_pred_exp	Request_Exp**	式ノードへのポインター。

戻り値 戻りコード。0 は成功を示します。

add_predicate 関数

目的 リストに述部を追加します。

構文

```
sqlint32 add_predicate (int a_pred_handle)
```

入力引き数

表 304. add_predicate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_pred_handle	int	述部式のハンドル。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

add_applied_predicate 関数

目的 適用済み述部リストに述部式を追加します。

構文

```
sqlint32 add_applied_predicate (int a_applied_pred_handle)
```

入力引き数

表 305. add_applied_predicate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_applied_pred_handle	int	適用済み述部式のハンドル。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 144 ページの『C++ API の要求クラス』

C++ API のデータ・クラス

以下の表で、C++ API の各データ・クラスについて説明します。

表 306. データ・クラス

クラス名	説明
Runtime_Data_Desc	データ項目の記述をカプセル化するクラス。
Runtime_Data_Desc_List	Runtime_Data_Desc のリスト。
Runtime_Data	ラッパーと照会ゲートウェイ・インターフェース間で渡されるデータ値をカプセル化するクラス。このデータ値は、結果となる列データ、または照会パラメーター内の列データのいずれかです。
Runtime_Data_List	Runtime_Data オブジェクトへのポインターのリスト。

関連情報:

- 188 ページの『Runtime_Data_Desc クラス』
- 192 ページの『Runtime_Data_Desc_List クラス』
- 195 ページの『Runtime_Data クラス』
- 203 ページの『Runtime_Data_List クラス』

Runtime_Data_Desc クラス

このセクションでは、Runtime_Data_Desc クラスについて説明し、その各メンバー関数について解説します。

概要

Runtime_Data_Desc クラスは、データ項目の記述をカプセル化します。

Runtime_Data_Desc クラスは、C++ API のデータ・クラスの 1 つです。

使用法 このクラスは DB2 照会ゲートウェイによりインスタンス化され、ラッパーによりサブクラス化されることはありません。このクラスは、パススルー操作の準備中に、ラッパーによりインスタンス化される場合があります。

ファイル

sqlqg_runtime_data_operation.h

データ・メンバー

なし。

コンストラクターおよびメンバー関数

以下の表で、Runtime_Data_Desc クラスのコンストラクターおよびメンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 307. Runtime_Data_Desc クラスのコンストラクター

コンストラクター	説明
Runtime_Data_Desc	Runtime_Data_Desc のインスタンスを構成します。

表 308. Runtime_Data_Desc クラスのメンバー関数

メンバー関数	説明
get_for_bit_data	データの FOR BIT DATA フラグを戻します。
get_null_indicator	NULL 可能標識を戻します。
get_data_type	データ・タイプを戻します。
get_maximum_length	タイプの最大長を戻します。

表 308. Runtime_Data_Desc クラスのメンバー関数 (続き)

メンバー関数	説明
get_precision	数値タイプの精度を戻します。
get_scale	数値タイプの尺度を戻します。
get_codepage	文字タイプのコード・ページを戻します。

Runtime_Data_Desc コンストラクター

目的 Runtime_Data_Desc のインスタンスを構成します。このコンストラクターは、パススルー・セッションの準備または記述操作に応じて Runtime_Data_Desc クラスをインスタンス化する際に使用されます。

構文

```
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)
```

入力引き数

表 309. Runtime_Data_Desc コンストラクターの入力引き数

資料名	データ・タイプ	説明
a_type	short	データ・タイプ。sql.h ヘッダー・ファイルに定義された SQL_TYP_xxx 値を使用します。
a_max_length	int	データの最大長。
a_codepage	short	文字データのコード・ページ。
a_null_ind	short	NULL 標識。
a_data_precision	unsigned char	数値データの精度。
a_data_scale	unsigned char	数値データ・タイプの尺度。
a_data_name	sqluint8*	データ項目名。
a_name_length	short	名前の長さ。

表 309. Runtime_Data_Desc コンストラクターの入力引き数 (続き)

資料名	データ・タイプ	説明
a_remote_type	short	リモート・タイプ・コード。

出力引き数

表 310. Runtime_Data_Desc コンストラクターの出力引き数

名前	データ・タイプ	説明
a_rc	sqlint32*	戻りコード値へのポインター。0 は成功を示します。

get_for_bit_data 関数

目的 データの FOR BIT DATA フラグを戻します。

構文

```
unsigned char get_for_bit_data ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 FOR BIT DATA フラグ (Y または N)。

get_null_indicator 関数

目的 NULL 可能標識を戻します。

構文

```
short get_null_indicator ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 NULL 標識フラグ。SQL_NULLABLE または SQL_NO_NULLS。

get_data_type 関数

目的 データ・タイプを戻します。

構文

```
short get_data_type ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 データ・タイプ (SQL_TYP_xxx)。SQL_TYP_xxx 記号については、sql.h ヘッダー・ファイルを参照。

get_maximum_length 関数

目的 タイプの最大長を戻します。

構文

```
int get_maximum_length ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 タイプの最大長。

get_precision 関数

目的 数値タイプの精度を戻します。

構文

```
unsigned char get_precision ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数値精度。

get_scale 関数

目的 数値タイプの尺度を戻します。

構文

```
unsigned char get_scale ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数値尺度。

get_codepage 関数

目的 文字タイプのコード・ページを戻します。

構文

```
unsigned short get_codepage ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 コード・ページ。

関連情報:

- 187 ページの『C++ API のデータ・クラス』

Runtime_Data_Desc_List クラス

このセクションでは、Runtime_Data_Desc_List クラスについて説明し、その各メンバー関数について解説します。

概要

Runtime_Data_Desc_List クラスは、Runtime_Data_Desc クラスのリストです。

Runtime_Data_Desc_List クラスは、C++ API のデータ・クラスの 1 つです。

使用法 このクラスは DB2 照会ゲートウェイによりインスタンス化されますが、ラッパーによりサブクラス化されることはありません。

ファイル
sqlqg_runtime_data_operation.h

データ・メンバー
なし。

メンバー関数

以下の表で、Runtime_Data_Desc_List クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 311. Runtime_Data_Desc_List クラスのメンバー関数

メンバー関数	説明
get_number_of_values	値の数を戻します。
set_number_of_values	リスト内の値の数を設定します。
get_ith_value	*i* 番目のデータ記述子を戻します。
set_ith_value	Runtime_Data_Desc ポインターを、リストの *i* 番目の位置に保管します。
operator[]	リスト内の *i* 番目の項目を、添え字表記を使用して戻します。

get_number_of_values 関数

目的 値の数を戻します。

構文

```
int get_number_of_values ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 値の数。

set_number_of_values 関数

目的 リスト内の値の数を設定します。このメンバー関数は 0 を設定することでリストを空にすることができます。また、リストを延長または短縮できます。

構文

```
sqlint32 set_number_of_values (int a_count)
```

入力引き数

表 312. set_number_of_values メンバー関数の入力引き数

名前	データ・タイプ	説明
a_count	int	値の数。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_ith_value 関数

目的 *i* 番目のデータ記述子を戻します。

構文

```
Runtime_Data_Desc* get_ith_value (int a_index)
```

入力引き数

表 313. get_ith_value メンバー関数の入力引き数

名前	データ・タイプ	説明
a_index	int	リストの索引 (0 から開始)。

出力引き数

なし。

戻り値 *i* 番目の値 (または NULL) へのポインター。

set_ith_value 関数

目的 Runtime_Data_Desc ポインターを、リストの *i* 番目の位置に保管します。

構文

```
sqlint32 set_ith_value (Runtime_Data_Desc* a_desc,
                        int                a_index)
```

入力引き数

表 314. set_ith_value メンバー関数の入力引き数

名前	データ・タイプ	説明
a_desc	Runtime_Data_Desc*	記述子ポインター。
a_index	int	索引 (0 から開始)。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

operator[] 関数

目的 リスト内の *i* 番目の項目を、添え字表記を使用して戻します。

構文

```
Runtime_Data_Desc* operator[] (int a_index)
```

入力引き数

表 315. operator[] メンバー関数の入力引き数

名前	データ・タイプ	説明
a_index	int	リストの索引 (0 から開始)。

出力引き数
なし。

戻り値 *i* 番目の項目。

関連情報:

- 187 ページの『C++ API のデータ・クラス』

Runtime_Data クラス

このセクションでは、Runtime_Data クラスについて説明し、その各メンバー関数について解説します。

概要

Runtime_Data クラスは、ラッパーと照会ゲートウェイ・インターフェース間で渡されるデータ値をカプセル化します。このデータ値は、結果となる列データ、または照会パラメーター内の列データのいずれかです。

Runtime_Data クラスは、C++ API のデータ・クラスの 1 つです。

使用法 このクラスは DB2 照会ゲートウェイによりインスタンス化され、ラッパーによりサブクラス化されることはありません。

ファイル
sqlqg_runtime_data_operation.h

データ・メンバー
なし。

メンバー関数

以下の表で、Runtime_Data クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 316. Runtime_Data クラスのメンバー関数

メンバー関数	説明
get_actual_length	データ値の実際の長さを戻します。
set_actual_length	データ値の実際の長さを設定します。
get_data	データ値へのポインターを戻します。
set_data	データ値に情報をコピーします。
is_data_null	データ値が NULL かどうか示します。
set_data_null	データ値を NULL としてマークします。
clear_null_indicator	データ値に対する NULL 標識をリセットします。
set_friendly_div_by_0	0 除算エラーが発生したため、値が NULL であることを示します。
set_friendly_exception	数値例外のため、値が NULL であることを示します。
is_data_nullable	データ値が NULL 可能かどうかの標識を戻します。
is_semantic_null	値が NULL を意味する標識を戻します。
check_friendly_div_by_0	NULL 標識の理由が 0 除算エラーであることを識別します。
check_friendly_exception	NULL 標識の理由が数値例外であることを識別します。
get_for_bit_data	データの FOR BIT DATA フラグを戻します。
get_null_indicator	NULL 可能標識を戻します。
get_data_type	データ・タイプを戻します。
get_maximum_length	最大長を戻します。
get_precision	数値タイプの精度を戻します。
get_scale	数値タイプの尺度を戻します。
get_codepage	文字タイプのコード・ページを戻します。

get_actual_length 関数

目的 データ値の実際の長さを戻します。

構文

```
int get_actual_length ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 実際のデータ長。

set_actual_length 関数

目的 データ値の実際の長さを設定します。

構文

```
void set_actual_length (int a_length)
```

入力引き数

表 317. set_actual_length メンバー関数の入力引き数

名前	データ・タイプ	説明
a_length	int	長さ。

出力引き数

なし。

戻り値 なし。

get_data 関数

目的 データ値へのポインターを戻します。

構文

```
unsigned char* get_data ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 データ値へのポインター。

set_data 関数

目的 データ値に情報をコピーします。

構文

```
sqlint32 set_data (unsigned char* a_data_ptr,
                   int            a_copy_len)
```

入力引き数

表 318. set_data メンバー関数の入力引き数

名前	データ・タイプ	説明
a_data_ptr	unsigned char*	データ値。
a_copy_len	int	値の長さ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

is_data_null 関数

目的 データ値が NULL かどうか示します。

構文

```
sqlint32 is_data_null ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 NULL 標識。

set_data_null 関数

目的 データ値を NULL としてマークします。

構文

```
sqlint32 set_data_null ()
```

入力引き数

なし。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

clear_null_indicator 関数

目的 データ値に対する NULL 標識をリセットします。

構文

```
sqlint32 clear_null_indicator ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

set_friendly_div_by_0 関数

目的 0 除算エラーが発生したため、値が NULL であることを示します。

構文

```
sqlint32 set_friendly_div_by_0 ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

set_friendly_exception 関数

目的 数値例外のため、値が NULL であることを示します。

構文

```
sqlint32 set_friendly_exception ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 戻りコード。0 は成功を示します。

is_data_nullable 関数

目的 データ値が NULL 可能かどうかの標識を戻します。

構文

```
sqlint32 is_data_nullable ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 NULL 可能標識。NULL 可能である場合は TRUE。

is_semantic_null 関数

目的 値が NULL を意味する標識を戻します。

構文

```
sqlint32 is_semantic_null ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 NULL 可能標識。NULL 可能である場合は TRUE。

check_friendly_div_by_0 関数

目的 NULL 標識の理由が 0 除算エラーであることを識別します。

構文

```
sqlint32 check_friendly_div_by_0
```

入力引き数
なし。

出力引き数
なし。

戻り値 NULL 標識。0 除算状態が発生した場合は TRUE。

check_friendly_exception 関数

目的 NULL 標識の理由が数値例外であることを識別します。

構文

```
sqlint32 check_friendly_exception ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 NULL 標識。例外が発生した場合は TRUE。

get_for_bit_data 関数

目的 データの FOR BIT DATA フラグを戻します。

構文

```
unsigned char get_for_bit_data ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 FOR BIT DATA フラグ (Y または N)。

get_null_indicator 関数

目的 NULL 可能標識を戻します。

構文

```
short get_null_indicator ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 NULL 標識フラグ。SQL_NULLABLE または SQL_NO_NULLS。

get_data_type 関数

目的 データ・タイプを戻します。

構文

```
short get_data_type ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 データ・タイプ (SQL_TYP_xxx)。SQL_TYP_xxx 記号については、sql.h ヘッダー・ファイルを参照。

get_maximum_length 関数

目的 タイプの最大長を戻します。

構文

```
int get_maximum_length ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 タイプの最大長。

get_precision 関数

目的 数値タイプの精度を戻します。

構文

```
unsigned char get_precision ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 数値精度。

get_scale 関数

目的 数値タイプの尺度を戻します。

構文

```
unsigned char get_scale ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 数値尺度。

get_codepage 関数

目的 文字タイプのコード・ページを戻します。

構文

```
unsigned short get_codepage ()
```

入力引き数
なし。

出力引き数
なし。

戻り値 コード・ページ。

関連情報:

- 187 ページの『C++ API のデータ・クラス』

Runtime_Data_List クラス

このセクションでは、Runtime_Data_List クラスについて説明し、その各メンバー関数について解説します。

概要

Runtime_Data_List クラスは、Runtime_Data オブジェクトへのポインターのリストです。

Runtime_Data_List クラスは、C++ API のデータ・クラスの 1 つです。

使用法 このクラスは DB2 照会ゲートウェイによりインスタンス化され、ラッパーによりサブクラス化されることはありません。

ファイル
sqlqg_runtime_data_operation.h

データ・メンバー
なし。

メンバー関数

以下の表で、Runtime_Data_List クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 319. Runtime_Data_List クラスのメンバー関数

メンバー関数	説明
get_number_of_values	値の数を戻します。
get_ith_value	*i* 番目のデータ記述子を戻します。
operator[]	リスト内の *i* 番目の項目を、添え字表記を使用して戻します。

get_number_of_values 関数

目的 値の数を戻します。

構文

```
int get_number_of_values ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 値の数。

get_ith_value 関数

目的 *i* 番目のデータ値を戻します。

構文

```
Runtime_Data* get_ith_value (int a_index)
```

入力引き数

表 320. get_ith_value メンバー関数の入力引き数

名前	データ・タイプ	説明
a_index	int	リストの索引 (0 から開始)。

出力引き数

なし。

戻り値 *i* 番目の値 (または NULL) へのポインター。

operator[] 関数

目的 リスト内の *i* 番目の項目を、添え字表記を使用して戻します。

構文

```
Runtime_Data* operator[] (int a_index)
```

入力引き数

表 321. operator[] メンバー関数の入力引き数

名前	データ・タイプ	説明
a_index	int	リストの索引 (0 から開始)。

出力引き数
なし。

戻り値 *i* 番目の項目。

関連情報:

• 187 ページの『C++ API のデータ・クラス』

Wrapper_Utilities クラス

このセクションでは、Wrapper_Utilities クラスについて説明し、その各メンバー関数について解説します。

概要

Wrapper_Utilities クラスは、複数の静的ユーティリティー関数のコンテナーです。Wrapper_Utilities クラスはインスタンス化またはサブクラス化しないでください。

Wrapper_Utilities クラスは C++ API のユーティリティー・クラスです。

ファイル
sqlqg_utils.h

データ・メンバー
なし。

メンバー関数

以下の表で、Wrapper_Utilities クラスの各メンバー関数について説明します。表の下に、それぞれの詳細を記載します。

表 322. Wrapper_Utilities クラスのメンバー関数

メンバー関数	説明
convert_to_upper	特定のコード・ページを使用し、文字ストリングを大文字に変換します。
convert_to_lower	特定のコード・ページを使用し、文字ストリングを小文字に変換します。
report_error	ユーザーにレポートするエラーを生成します。
report_warning	ユーザーにレポートする警告を生成します。
allocate	メモリーのブロックを割り振ります。
deallocate	allocate() を使用して割り振られたメモリーのブロックを解放します。
get_sb_DB_codepage	現行データベースの単一バイト・コード・ページを戻します。
get_db_DB_codepage	現行データベースの 2 バイト・コード・ページを戻します。
string_to_tokens	文字ストリングをスキャンし、ストリングを連続するトークンに分割します。
get_db2_install_path	DB2 Information Integrator のインストール・ディレクトリーの絶対パス名を示す、ヌル終了文字ストリングを戻します。
get_db2_instance_path	DB2 Information Integrator のインスタンスの絶対パス名を示す、ヌル終了文字ストリングを戻します。
trace_data	DB2 トレース機能へ情報ブロックを書き込みます。

convert_to_upper 関数

目的 特定のコード・ページを使用し、文字ストリングを大文字に変換します。

使用法 get_sb_DB_codepage() または get_db_DB_codepage() 関数を使用して、現行データベースのコード・ページを取得します。

構文

```
int convert_to_upper (char*        a_string,
                      size_t       a_length,
                      unsigned int a_codepage)
```

入力引き数

表 323. convert_to_upper メンバー関数の入力引き数

名前	データ・タイプ	説明
a_string	char*	変換するストリング (非ヌル終了)。
a_length	size_t	ストリングの長さ。
a_codepage	unsigned int	変換用のコード・ページ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。-1 は、コード・ページが定義されておらず、変換対象のストリングに外字が含まれることを示します。これらの外字はスキップされます。

convert_to_lower 関数

目的 特定のコード・ページを使用し、文字ストリングを小文字に変換します。

使用法 get_sb_DB_codepage() または get_db_DB_codepage() 関数を使用して、現行データベースのコード・ページを取得します。

構文

```
int convert_to_lower (char*        a_string,
                      size_t       a_length,
                      unsigned int a_codepage)
```

入力引き数

表 324. convert_to_lower メンバー関数の入力引き数

名前	データ・タイプ	説明
a_string	char*	変換するストリング (非ヌル終了)。
a_length	size_t	ストリングの長さ。
a_codepage	unsigned int	変換用のコード・ページ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。-1 は、コード・ページが定義されておらず、変換対象のストリングに外字が含まれることを示します。これらの外字はスキップされます。

report_error 関数

目的 ユーザーにレポートするエラーを生成します。

構文

```
sqlint32 report_error (const char* a_function_name,
                       int         a_sql_code,
                       int         a_number_of_tokens,
                         ...)
```

入力引き数

表 325. report_error メンバー関数の入力引き数

名前	データ・タイプ	説明
a_function_name	const char*	エラーを生成する関数を示すヌル終了文字ストリング。最大長は 5 文字です。このストリング値は SQLCA の sqlerrp フィールドを介してクライアント・プログラムにアクセス可能であり、SQL の接頭部付きの大文字で表示されます。
a_sql_code	int	エラーの事前定義された SQL コード。sqlcodes.h ファイルを参照。
a_number_of_tokens	int	メッセージの置換トークンの数。
...	(int, const char*)	トークンごとに 1 対の値。各ペアは、整数の長さと、非ヌル終了の文字ストリングへのポインターから構成されます。

出力引き数

なし。

戻り値 戻りコード。この関数は呼び出し元に報告されたエラー・コードを戻し、呼び出し元がこのエラー・コードを DB2 に戻します。

report_warning 関数

目的 ユーザーにレポートする警告を生成します。

構文

```
sqlint32 report_warning (const char*  a_function_name,
                         int          a_warning_index,
                         char         a_warning_flag,
                         int          a_number_of_tokens,
                           ...)
```

入力引き数

表 326. report_warning メンバー関数の入力引き数

名前	データ・タイプ	説明
a_function_name	const char*	警告を生成する関数を示すヌル終了文字ストリング。最大長は 5 文字です。このストリング値は SQLCA の sqlerrp フィールドを介してクライアント・プログラムにアクセス可能であり、SQL の接頭部付きの大文字で表示されます。
a_warning_index	int	警告のタイプを示す値。sql.h ヘッダー・ファイルに定義された SQL_WARN_xxx 定数を使用します。
a_warning_flag	char	警告標識。sql.h ヘッダー・ファイルに定義された SQL_WARNING 定数を使用します。
a_number_of_tokens	int	SQLCA の sqlerrm フィールドを介して指定され、クライアント・プログラムにアクセス可能なトークン (長さとストリングのペア) の数。
...	(int, const char*)	トークンごとに 1 対の値。各ペアは、整数の長さと、非ヌル終了の文字ストリングへのポインターから構成されます。

出力引き数

なし。

戻り値 0 は警告が正常に生成されたことを示します。 0 以外の値は、警告の生成中に問題が発生したことを示します。 0 以外の戻りコードは、DB2 に戻される必要があります。

allocate 関数

目的 メモリーのブロックを割り振ります。

構文

```
int allocate (size_t a_size,
              void** a_block)
```

入力引き数

表 327. allocate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_size	size_t	割り振るブロックのサイズ。

出力引き数

表 328. allocate メンバー関数の出力引き数

名前	データ・タイプ	説明
a_block	void**	割り振られたブロックへのポインター。

戻り値 戻りコード。0 は成功を示します。

deallocate 関数

目的 allocate() を使用して割り振られたメモリーのブロックを解放します。

構文

```
void deallocate (void* a_block)
```

入力引き数

表 329. deallocate メンバー関数の入力引き数

名前	データ・タイプ	説明
a_block	void*	解放するメモリー・ブロック。

出力引き数

なし。

戻り値 なし。

get_sb_DB_codepage 関数

目的 現行データベースの単一バイト・コード・ページを戻します。

構文

```
int get_sb_DB_codepage ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 コード・ページ。

get_db_DB_codepage 関数

目的 現行データベースの 2 バイト・コード・ページを戻します。

構文

```
int get_db_DB_codepage ()
```

入力引き数

なし。

出力引き数

なし。

戻り値 コード・ページ。

string_to_tokens 関数

目的 文字ストリングをスキャンし、ストリングを連続するトークンに分割します。このメンバー関数は、strtok() 関数および strtok_r() 関数の代替として使用できる、安全性の高い関数です。

構文

```
char* string_to_tokens (char*       a_string,
                        const char* a_sep,
                        char**      a_last)
```

入力引き数

表 330. string_to_tokens メンバー関数の入力引き数

名前	データ・タイプ	説明
a_string	char*	スキャンするストリング。

表 330. string_to_tokens メンバー関数の入力引き数 (続き)

名前	データ・タイプ	説明
a_sep	const char*	トークン間の区切り記号として使用されるストリング。
a_last	char*	string_to_tokens の呼び出し間の状態保存に使用する値。

出力引き数

なし。

戻り値 ストリング内の次のトークンへのポインターまたは NULL。

get_db2_install_path 関数

目的 DB2 Information Integrator のインストール・ディレクトリーの絶対パス名を示す、ヌル終了文字ストリングを戻します。

構文

```
sqlint32 get_db2_install_path (char*    a_path,
                               sqlint32 a_path_size)
```

入力引き数

表 331. get_db2_install_path メンバー関数の入力引き数

名前	データ・タイプ	説明
a_path	char*	パス名を含むバッファーへのポインター。
a_path_size	sqlint32	バッファーの長さ (NULL 終止符のスペースを含む)。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

get_db2_instance_path 関数

目的 DB2 Information Integrator のインスタンスの絶対パス名を示す、ヌル終了文字ストリングを戻します。

構文

```
sqlint32 get_db2_instance_path (char*    a_path,
                                sqlint32 a_path_size)
```

入力引き数

表 332. get_db2_instance_path メンバー関数の入力引き数

名前	データ・タイプ	説明
a_path	char*	パス名を含むバッファーへのポインター。
a_path_size	sqlint32	バッファーの長さ (NULL 終止符のスペースを含む)。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

trace_data 関数

目的 DB2 トレース機能へ情報ブロックを書き込みます。

使用法 DB2 トレース機能の詳細については、「*DB2 コマンド・リファレンス*」を参照してください。

構文

```
void trace_data (int    a_probe,
                 void*  a_data,
                 int    a_data_size)
```

入力引き数

表 333. trace_data メンバー関数の入力引き数

名前	データ・タイプ	説明
a_probe	int	トレース・ポイントを示す数値。
a_data	void*	トレースするデータへのポインター。
a_data_size	int	トレースするデータの長さ。

出力引き数

なし。

戻り値 戻りコード。0 は成功を示します。

関連情報:

- 1 ページの『C++ API のカタログ・クラス』
- 187 ページの『C++ API のデータ・クラス』

DB2 Information Integrator の技術資料

以下のいくつかのトピックでは、次の実行方法を説明します。

- 資料およびリリース情報にアクセスする (資料の印刷および注文も含む)
- DB2 Information Integrator インフォメーション・センターまたは DB2 HTML ドキュメンテーション CD を使用してトピックにアクセスする

資料およびリリース情報へのアクセス

DB2 Information Integrator の技術情報は、以下の形式で利用可能です。

- ブック (PDF およびハードコピー)。DB2 Information Integrator ライブラリー内の各ブックの説明は、 www.ibm.com/shop/publications/order の IBM Publications Center にあります。
- インフォメーション・センター (HTML 形式)。
- DB2 データベース・ツールのヘルプ (HTML 形式)。

DB2 Information Integrator の資料

DB2 Information Integrator PDF ドキュメンテーション CD には、 DB2 Information Integrator ライブラリーと DB2 Universal Database ライブラリーの資料の PDF ファイルが収録されています。 DB2 Information Integrator PDF ドキュメンテーション CD の構造は、以下のようになっています。

- Windows オペレーティング・システムの場合: *x*:¥doc¥*%L*
- UNIX オペレーティング・システムの場合: */cdrom*/doc/*%L/*

各部分の意味は以下のとおりです。

- *x* は Windows CD-ROM ドライブ名を表します。
- *cdrom* は CD-ROM のマウント・ポイントを表します (UNIX)。
- *%L* は使用する資料のロケールです (en_US など)。

言語	ロケール	ID	言語	ロケール	ID
アラビア語	ar_AA	w	日本語	ja_JP	j
ブラジル・ポルトガル語	pt_BR	b	韓国語	ko_KR	k
ブルガリア語	bg_BG	u	ノルウェー語	no_NO	n
クロアチア語	hr_HR	9	ポーランド語	pl_PL	p
チェコ語	cs_CZ	x	ポルトガル語	pt_PT	v
デンマーク語	da_DK	d	ルーマニア語	ro_RO	8
オランダ語	nl_NL	q	ロシア語	ru_RU	r
英語	en_US	e	中国語 (簡体字)	zh_CN	c

フィンランド語	fi_FI	y	スロバキア語	sk_SK	7
フランス語	fr_FR	f	スロベニア語	sl_SI	l
ドイツ語	de_DE	g	スペイン語	es_ES	z
ギリシャ語	el_GR	a	スウェーデン語	sv_SE	s
ハンガリー語	hu_HU	h	中国語 (繁体字)	zh_TW	t
イタリア語	it_IT	i	トルコ語	tr_TR	m

各 PDF ファイル名の 6 番目の文字は資料の言語を示しています (以下の表をご覧ください)。たとえば、ファイル名 `iiyige80` は「*IBM DB2 Information Integrator インストール・ガイド*」の英語版を示しており、ファイル名 `iiyigg80` は同じ資料のドイツ語版を示しています。

以下の表に、DB2 Information Integrator 用の利用可能な資料を示します。

表 334. DB2 Information Integrator 資料

資料名	資料番号	インストール・カテゴリー	PDF ファイル名
「*IBM DB2 Information Integrator ソリューション・ガイド*」	SC88-9563	getting_started	iiyisj80
「*IBM DB2 Information Integrator インストール・ガイド*」	GC88-9562	getting_started	iiyigj80
「*IBM DB2 Information Integrator マイグレーション・ガイド*」	SC88-9610	getting_started	iiymgj80
「*IBM DB2 Information Integrator 連合システム・ガイド*」	SC88-9164	admin	iiyfpj80
「*IBM DB2 Information Integrator データ・ソース構成ガイド*」	オンラインでのみ入手可能	optional	iiylsj80
「*IBM DB2 Information Integrator 開発者向けガイド*」	SC88-9609	ad	iiyfsj80

PDF ファイルからの資料の印刷方法

DB2 Information Integrator PDF ドキュメンテーション CD に収録されている DB2 Information Integrator 資料の PDF ファイルを印刷することができます。 Adobe Acrobat Reader を使用して、資料全体、一定範囲のページ、または特定のページを印刷できます。

前提条件:

Adobe Acrobat Reader がインストールされていることを確認してください。このプログラムは、Adobe 社の Web サイト (www.adobe.com) から入手できます。

手順:

PDF ファイルから DB2 Information Integrator 資料を印刷するには以下のようにします。

1. DB2 Information Integrator PDF ドキュメンテーション CD をドライブに挿入します。UNIX オペレーティング・システムの場合、CD をマウントします。
2. Adobe Acrobat Reader を起動します。
3. 以下に示すいずれかの場所から PDF ファイルを開きます。
   - Windows オペレーティング・システムの場合: *x*:¥doc¥*%L*
   - UNIX オペレーティング・システムの場合: */cdrom*/doc/*%L*/

   各部分の意味は以下のとおりです。
   - *x* は Windows CD-ROM ドライブ名を表します。
   - *cdrom* は CD-ROM のマウント・ポイントを表します (UNIX)。
   - *%L* は印刷する資料のロケールです (en_US など)。
4. **「ファイル」->「印刷」**をクリックします。
5. 「印刷」ウィンドウで、すべてのページを印刷するか、現在のページを印刷するか、ページ範囲を印刷するかを指定します。
6. **「OK」**をクリックします。

印刷資料の注文方法

ご使用の DB2 Information Integrator 製品用の Doc Pack (ドキュメンテーション・パッケージ) を IBM 販売店に注文することによって、印刷された DB2 Information Integrator マニュアルを入手できます。 Doc Pack とは、DB2 Information Integrator ライブラリーのマニュアルのサブセットです。 Doc Pack は、購入した DB2 製品を使い始めるのに役立つように作られています。

Doc Pack のマニュアルは、 DB2 Information Integrator 製品に付属の DB2 Information Integrator PDF ドキュメンテーション CD に収録されているマニュアルと同じものです。

以下の方法で、個々の資料を注文することもできます。

- IBM 営業担当員または認定販売業者に連絡する。お客様がお住まいの地域の IBM 担当員の情報については、お手数ですが IBM の Web サイト (www.ibm.com/planetwide) の IBM Worldwide Directory of Contacts で確認してください。
- IBM Publications Center (www.ibm.com/shop/publications/order) にアクセスする。

リリース情報およびインストール要件

リリース情報およびインストール要件では、製品のリリースおよびフィックスパックのレベルに固有の情報を提供します。リリース情報は、各リリースおよびフィックスパックに組み込まれているドキュメンテーション更新の要約も提供します。

リリース情報およびインストール要件は、テキスト形式および HTML 形式で製品 CD-ROM から入手できます。

- Windows オペレーティング・システムの場合: *x*:¥doc¥*%L*
- UNIX オペレーティング・システムの場合: */cdrom*/doc/*%L*/

各部分の意味は以下のとおりです。

- *x* は Windows CD-ROM ドライブ名を表します。
- *cdrom* は CD-ROM のマウント・ポイントを表します (UNIX)。
- *%L* は使用する資料のロケールです (en_US など)。

表 335. リリース情報

資料名	ファイル名	位置
「*DB2 Information Integrator リリース情報*」	ReleaseNotes	• 製品 CD-ROM • DB2 Information Integrator インフォメーション・センター • DB2 Information Integration Installation Launchpad
「*DB2 Information Integrator インストール要件*」	Prereqs	• 製品 CD-ROM • DB2 Information Integration Installation Launchpad

DB2 Information Integrator ドキュメンテーション・フィックスパック

IBM は定期的にドキュメンテーション・フィックスパックを提供しています。ドキュメンテーション・フィックスパックによって、新しい情報が入手可能になった場合に、DB2 HTML ドキュメンテーション CD からインストールした情報をアップデートすることができます。

ドキュメンテーション・フィックスパックは累積的なものです。たとえば、バージョン 8.1 用のドキュメンテーションをインストールしてからバージョン 8.1.2 を適用すると、フィックスパック 1 のアップデートとバージョン 8.1.2 のアップデートがドキュメンテーションに適用されます。

ドキュメンテーション・フィックスパックをインストールすると、HTML ドキュメンテーションには、製品の印刷版マニュアルおよびオンライン PDF マニュアルよりも新しい情報が記載されることになります。

DB2 Information Integrator インフォメーション・センターまたは DB2 HTML ドキュメンテーション CD を使ったトピックへのアクセス

DB2 Information Integrator インフォメーション・センターを使用すると、DB2 Information Integrator をビジネスで活用するために必要な情報にアクセスできます。

DB2 Information Integrator インフォメーション・センターの機能

DB2 Information Integrator インフォメーション・センターには、以下の機能があります。

統合されたナビゲーション・ツリー
1 つのナビゲーション・ツリーから、DB2 Information Integrator ライブラリー内のすべてのトピックを探し出すことができます。

検索 ナビゲーション・ツールバーの**「検索」**をクリックすることによって、ワークステーション上のすべてのトピックを検索できます。

マスター索引
1 つのマスター・インデックスからトピックや Tools ヘルプの情報にアクセスできます。この索引には、DB2 Information Integrator ライブラリー全体の項目が入っています。

マスター用語集
マスター用語集は、DB2 Information Integrator ライブラリーで使用されている用語を定義しています。

資料の定期的な更新
更新された HTML トピックをダウンロードすることによって、トピックを最新の状態に保つことができます。

DB2 Information Integrator インフォメーション・センターのトピックを見つける

DB2 Information Integrator インフォメーション・センターは、次の主なエレメントから構成されています。

ナビゲーション・ツリー
ナビゲーション・ツリーは、ブラウザー・ウィンドウの左側のフレームにあり

ます。ツリーを拡張したり縮小したりして、DB2 Information Integrator インフォメーション・センター内のトピック・リンクや、用語集、マスター索引などを表示したり、隠したりできます。

ナビゲーション・ツールバー

ナビゲーション・ツールバーは、ブラウザー・ウィンドウの上部右側のフレームにあります。ナビゲーション・ツールバー各種のプッシュボタンを使って、DB2 Information Integrator インフォメーション・センターを検索したり、ナビゲーション・ツリーを隠したり、ナビゲーション・ツリーに現在表示されているトピックを検索したりできます。

内容フレーム

内容フレームは、ブラウザー・ウィンドウの下部右側のフレームにあります。ナビゲーション・ツリーのリンクをクリックしたり、検索結果をクリックしたり、別のトピックやマスター索引からのリンクをクリックしたりすると、内容フレームに該当するトピックが表示されます。

前提条件:

ブラウザーから DB2 Information Integrator インフォメーション・センターにアクセスするには、以下のいずれかのブラウザーを使用する必要があります。

- Microsoft Explorer バージョン 5 以上
- Netscape Navigator バージョン 6.1 以上

制約事項:

DB2 Information Integrator インフォメーション・センターには、DB2 HTML ドキュメンテーション CD から選択してインストールしたトピックだけが入っています。トピックへのリンクをクリックするときに、Web ブラウザーが「ファイルが見つかりません」というエラーを戻す場合、 DB2 HTML ドキュメンテーション CD から追加のトピックをインストールしてください。

手順:

キーワードを指定してトピックを検索するには以下のようにします。

1. ナビゲーション・ツールバーで、**「検索」**をクリックします。
2. 「検索」ウィンドウの一番上のテキスト入力フィールドに、参照したいトピックに関係する用語を 2 つ以上入力して、**「検索」**をクリックします。**「結果」**フィールドにトピックのリストが表示されます。入力した検索ストリングに最も近似のトピックがリストの最上部に表示されます。

   入力する用語が多いほど、検索の精度が上がり、結果のトピック数を減らすことができます。
3. **「結果」**フィールドで、 参照したいトピックのタイトルをクリックします。そうすると、そのトピックが内容フレームに表示されます。

ナビゲーション・ツリー内のトピックを検索するには以下のようにします。

1. ナビゲーション・ツリーで、参照したい内容に関係したトピックのカテゴリーのとなりのブック・アイコンをクリックします。サブカテゴリーのリストがアイコンの下に表示されます。
2. 参照したいトピックの入ったカテゴリーが見つかるまで、ブック・アイコンをクリックします。トピックにリンク接続されているカテゴリーは、そのカテゴリー・タイトルの上にカーソルを置くと、リンクとして表示されます。ナビゲーション・ツリーの中で、ページ・アイコンはトピックを識別するために使用されています。
3. トピック・リンクをクリックします。そうすると、そのトピックが内容フレームに表示されます。

マスター索引内のトピックを検索するには以下のようにします。

1. ナビゲーション・ツリーで、**「索引」**をクリックします。索引が拡張され、50 音順のリンクのリストが表示されます。
2. ナビゲーション・ツリーで、探しているサブジェクトの最初の文字をクリックします。その文字で始まる項目のリストが内容フレームに表示されます。ブック・アイコンが表示される場合、そのサブジェクトには複数の索引項目があります。
3. 参照したいサブジェクトに対応したブック・アイコンをクリックします。クリックした用語の下に、関係するトピックのリストが表示されます。
4. 参照したいトピックのリストをクリックします。そうすると、そのトピックが内容フレームに表示されます。

DB2 HTML ドキュメンテーションの使用

このトピックでは、DB2 HTML ドキュメンテーション CD にあるドキュメンテーションをインストール、表示、およびコピーする方法と、インストール後にそれらを更新する方法を説明しています。

DB2 HTML ドキュメンテーションのインストール

DB2 HTML ドキュメンテーション CD のインストール・ディレクトリーは、情報のカテゴリーごとに異なります。以下のとおりです。

htmlcdpath/doc/htmlcd/*%L*/*category*

htmlcdpath
: DB2 HTML ドキュメンテーション CD がインストールされるディレクトリーです。

%L
: 使用する資料のロケールです (en_US など)。

category
: カテゴリー ID。たとえば、インストール情報は getting_started です。

DB2 HTML ドキュメンテーション CD から技術資料を直接参照する

すべての HTML トピックは DB2 HTML ドキュメンテーション CD から表示できます。

制約事項:

オンライン・ヘルプを表示するには、DB2 製品をインストールすることが必要です。

手順:

DB2 HTML ドキュメンテーション CD から HTML ドキュメンテーションを表示するには、以下の手順に従います。

1. DB2 HTML ドキュメンテーション CD をドライブに挿入します。
2. Web ブラウザーを起動して、以下のファイルを開きます。
   - Windows オペレーティング・システムの場合
     x:¥Program Files¥IBM¥sqllib¥doc¥htmlcd¥*%L*¥index.*
   - UNIX オペレーティング・システムの場合
     /cdrom/Program Files/IBM/sqllib/doc/htmlcd/*%L*/index.htm

   各部分の意味は以下のとおりです。
   - *x* は Windows CD-ROM ドライブ名を表します。
   - *cdrom* は CD-ROM のマウント・ポイントを表します (UNIX)。
   - *%L* は使用する資料のロケールです (en_US など)。

DB2 HTML ドキュメンテーション CD から Web サーバーへファイルをコピーする

すべての DB2 ライブラリーは DB2 HTML ドキュメンテーション CD に収録されています。ライブラリーを有効に活用するため、Web サーバーにインストールすることができます。

手順:

ファイルを DB2 HTML ドキュメンテーション CD から Web サーバー上の適切なパスにコピーします (デフォルトのパスを以下に示します)。

- Windows オペレーティング・システムの場合: *x*:¥Program Files¥IBM¥sqllib¥doc¥htmlcd¥*%L*¥*.*
- UNIX オペレーティング・システムの場合: */cdrom*/Program Files/IBM/sqllib/doc/htmlcd/*%L*

各部分の意味は以下のとおりです。

- *x* は Windows CD-ROM ドライブ名を表します。
- *cdrom* は CD-ROM のマウント・ポイントを表します (UNIX)。
- *%L* は使用する資料のロケールです (en_US など)。

コンピューター上の HTML 資料をアップデートする

IBM がアップデートを提供した場合、DB2 HTML ドキュメンテーション CD からインストールした HTML ファイルを、以下のいずれかの方法でアップデートできます。

- インフォメーション・センターを使用する (DB2 管理グラフィカル・ユーザー・インターフェース・ツールをインストールしてある場合)。
- DB2 HTML ドキュメンテーション・フィックスパックをダウンロードして適用する。

これらの手順を実行しても、DB2 のコードがアップデートされるわけではありません。

前提条件:

必要な場合にはアップデート・プログラムが IBM サーバーから最新のドキュメンテーション・フィックスパックをダウンロードするので、コンピューターがインターネットに接続されていることを確認します。インターネットに接続するには、プロキシー情報を提供することが必要な場合もあります。

手順:

インフォメーション・センターを使用して、 マシンにインストールされている HTML ドキュメンテーションをアップデートするには以下のようにします。

1. DB2 インフォメーション・センターを開始します。
   - グラフィカル管理ツールから、ツールバーの**「インフォメーション・センター」**アイコンをクリックします。
   - コマンド行で、db2ic と入力します。
2. **「インフォメーション・センター」** -> **「Update Local Documentation (ローカル・ドキュメンテーションのアップデート)」**をクリックして、アップデートを開始します。

   ドキュメンテーション・アップデートが入手可能であれば、ダウンロードされて、適用されます。

手操作でドキュメンテーション・アップデートをダウンロードして適用するには、以下のようにします。

1. Web ブラウザーで DB2 サポート・ページ www.ibm.com/software/data/db2/udb/winos2unix/support を開きます。
2. **「DB2 Version 8」**をクリックして、ご使用のオペレーティング・システム用のドキュメンテーション・フィックスパックのリンクを探します。
3. ドキュメンテーション・フィックスパックのレベルと、マシンにインストールされている文書のレベルを比較して、ローカルな DB2 文書が更新されているかどうかを確認します。

4. さらに新しいバージョンの文書がある場合、ご使用のオペレーティング・システムに対応するフィックスパックをダウンロードします。すべての Windows オペレーティング・システム用に 1 つ、およびすべての UNIX オペレーティング・システムに 1 つのフィックスパックがあります。
5. 以下のようにしてフィックスパックを適用します。
   - Windows オペレーティング・システムの場合、フィックスパックは自己抽出型の zip ファイルとなっています。空のディレクトリーにドキュメンテーション・フィックスパックをダウンロードし、unzip します。そのディレクトリーで **setup** コマンドを実行して、ドキュメンテーション・フィックスパックをインストールします。
   - UNIX オペレーティング・システムの場合、ドキュメンテーション・フィックスパックは圧縮された tar.Z となっています。 tar ファイルが解凍・展開されると、`delta_install` というディレクトリーが作成されます。そのディレクトリー内でスクリプト installdocfix を実行して、ドキュメンテーション・フィックスパックをインストールします。

DB2 文書の検索

DB2 資料を検索するには、Netscape バージョン 6.1 以上または Microsoft Internet Explorer バージョン 5 以上を使用してください。ブラウザーの Java サポートが有効になっていることを確認してください。

ブラウザーから DB2 Information Integrator インフォメーション・センターにアクセスして、ナビゲーション・ツールバーの検索アイコンをクリックすると、検索ウィンドウが開きます。初めて検索機能を実行する場合、検索ウィンドウのロードに 1 分程度かかる場合があります。

制約事項:

文書検索を実行するときには、以下の制約事項があります。

- ブール検索はサポートされていません。ブール検索修飾子 *and* および *or* は、検索において無視されます。たとえば、以下の検索の結果は同じになります。
  - サーブレット *and* Bean
  - サーブレット *or* Bean
- ワイルドカード検索はサポートされていません。 *java** を検索しても、リテラル・ストリング *java** が検出され、 *javadoc* などは検出されません。

一般に、単一の語を検索するよりも句を検索したほうが、より良い検索結果が得られます。

手順:

DB2 文書を検索するには以下のようにします。

1. ナビゲーション・ツールバーで、**「検索」**をクリックします。
2. 「検索」ウィンドウの一番上のテキスト入力フィールドに、参照したいトピックに関係する用語を 2 つ以上入力して、**「検索」**をクリックします。**「結果」**フィールドに、正確さのランクに従ってトピックのリストが表示されます。

   入力する用語が多いほど、検索の精度が上がり、結果のトピック数を減らすことができます。
3. **「結果」**フィールドで、 参照したいトピックのタイトルをクリックします。そうすると、そのトピックが内容フレームに表示されます。

検索を実行するときには、最初の結果が自動的にブラウザー・フレームにロードされます。他の検索結果の内容を表示するには、リストから結果をクリックします。

Netscape 4.x を使って DB2 資料を検索する場合のトラブルシューティング

検索の問題のほとんどは、Web ブラウザーの Java サポートに関係しています。このトピックでは、可能な解決策を記載しています。

手順:

Netscape 4.x に共通している問題は、セキュリティー・クラスが欠落しているか、配置が間違っていることです。以下に示す解決策を実行してみてください (特にブラウザーの Java コンソールに以下の行がある場合は必ず実行してください)。

```
Cannot find class  java/security/InvalidParameterException
```

以下のファイルを、DB2 HTML ドキュメンテーション CD から、Netscape ブラウザーがインストールされているディレクトリー内にある `java¥classes¥java¥security¥` ディレクトリーにコピーします。`java¥security¥` サブディレクトリーを作成することが必要な場合もあります。

- Windows オペレーティング・システムの場合

  x`:Program Files¥IBM¥sqllib¥doc¥htmlcd¥%L¥InvalidParameterException.class`
- UNIX オペレーティング・システムの場合

  `/`*cdrom*`/Program Files/IBM/sqllib/doc/htmlcd/%L`
  `/InvalidParameterException.class`

各部分の意味は以下のとおりです。

- *x* は Windows CD-ROM ドライブ名を表します。
- *cdrom* は CD-ROM のマウント・ポイントを表します (UNIX)。
- *%L* は使用する資料のロケールです (en_US など)。

上記の対処策を実行しても Netscape ブラウザーで検索入力ウィンドウが表示されない場合は、以下の処置を実行してください。

- Netscape ブラウザーのすべてのインスタンスを停止して、コンピューター上で Netscape コードが実行されていないことを確認します。その後、Netscape ブラウザーの新しいインスタンスを開き、再度検索を開始します。
- ブラウザーのキャッシュを削除します。
- Netscape の別のバージョンまたは別のブラウザーを使用します。

アクセシビリティー

身体に障害のある (身体動作が制限されている、視力が弱いなど) ユーザーは、アクセシビリティー機能を使用することによって、ソフトウェア製品を十分活用できます。DB2 Information Integrator バージョン 8 に備わっている主なアクセシビリティー機能は以下のとおりです。

- すべての機能をマウスの代わりにキーボードを使用して操作できます。
- フォントのサイズと色をカスタマイズできます。
- アラートを表示にするか音声にするかを指定できます。
- DB2 は、Java™ Accessibility API を使用するアクセシビリティー・アプリケーションをサポートします。
- DB2 の資料は、アクセスしやすい形式で提供されています。

キーボードによる入力およびナビゲーション

キーボードだけを使用して、コントロール・センター、データウェアハウス・センター、レプリケーション・センターなどの、DB2 データベース・ツールを操作できます。マウスの代わりに複数のキーまたはキーの組み合わせを使用してほとんどの操作を実行できます。

UNIX ベースのシステムでは、キーボード・フォーカスの置かれている位置が強調表示されます。この強調表示によって、アクティブなウィンドウ領域が示されます。そのウィンドウ領域が、ユーザーのキー・ストロークの対象となります。

アクセスしやすい表示

DB2 データベース・ツールには、視力の弱いユーザーのためにユーザー・インターフェースを拡張し、アクセシビリティーを向上させる機能が備わっています。これらのアクセシビリティー拡張機能には、フォント・プロパティーのカスタマイズを可能にする機能も含まれています。

フォントの設定

DB2 データベース・ツールでは、「ツール設定」ノートブックを使用して、メニューおよびウィンドウに使用されるテキストの色、サイズ、およびフォントを選択できます。

色覚への非依存

この製品の機能を使用するために、ユーザーは色を識別する必要はありません。

代替アラート・キュー

「ツール設定」ノートブックを使用して、アラートを音声にするか、表示にするかを指定できます。

支援テクノロジーとの互換性

DB2 Information Integrator インターフェースは、身体に障害を持つ人々によって使用されているスクリーン・リーダーおよび他の支援テクノロジーに採用されている Java Accessibility API をサポートしています。

入手可能な資料

DB2 ファミリー製品の資料は HTML 形式で入手可能です。資料は、ご使用のブラウザーに設定されている表示設定に従って表示することができます。さらに、スクリーン・リーダーや他の支援テクノロジーを使用することもできます。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。使用許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。 国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。 さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。 お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プ

ログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (*お客様の会社名*) (*西暦年*). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _*年を入れる*_. All rights reserved.

商標

以下は、IBM Corporation の商標です。

IBM
DB2

以下は、それぞれ各社の商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[カ行]

カタログ・クラス
  リスト 1
  Catalog_Option 2
  Column_Info 30
  Nickname_Info 49
  Server_Info 13
  User_Info 23
  Wrapper_Info 4
コンストラクター
  Column_Info クラス 30
  Fenced_Generic_Nickname クラス 115
  Fenced_Generic_Server クラス 90
  Fenced_Generic_User クラス 103
  Fenced_Generic_Wrapper クラス 72
  Nickname_Info クラス 49
  Remote_Connection クラス 121
  Remote_Passthru クラス 138
  Remote_Query クラス 128
  Reply クラス 152
  Runtime_Data_Desc クラス 188
  Server_Info クラス 13
  Unfenced_Generic_Nickname クラス 107
  Unfenced_Generic_Server クラス 79
  Unfenced_Generic_User クラス 97
  Unfenced_Generic_Wrapper クラス 65
  User_Info クラス 23
  Wrapper_Info クラス 4

[サ行]

サーバー・クラス
  リスト 78
  Fenced_Generic_Server 90
  Unfenced_Generic_Server 79
操作クラス
  リスト 128
  Remote_Passthru 138
  Remote_Query 128

[タ行]

データ・クラス
  リスト 187
  Runtime_Data 195
  Runtime_Data_Desc 188
  Runtime_Data_Desc_List 192
  Runtime_Data_List 203
データ・メンバー
  Fenced_Generic_Nickname クラス 115
  Fenced_Generic_Server クラス 90
  Fenced_Generic_User クラス 103
  Remote_Connection クラス 121
  Unfenced_Generic_Nickname クラス 107
  Unfenced_Generic_Server クラス 79
  Unfenced_Generic_User クラス 97
  Unfenced_Generic_Wrapper クラス 65
デストラクター
  Fenced_Generic_User クラス 103
  Fenced_Generic_Wrapper クラス 72
  Unfenced_Generic_User クラス 97
  Unfenced_Generic_Wrapper クラス 65

[ナ行]

ニックネーム・クラス
  リスト 106
  Fenced_Generic_Nickname 115
  Unfenced_Generic_Nickname 107

[マ行]

メンバー関数
  Catalog_Option クラス 2
  Column_Info クラス 30
  Fenced_Generic_Nickname クラス 115
  Fenced_Generic_Server クラス 90
  Fenced_Generic_User クラス 103
  Fenced_Generic_Wrapper クラス 72
  Nickname_Info クラス 49
  Predicate_List クラス 181
  Remote_Connection クラス 121
  Remote_Passthru クラス 138
  Remote_Query クラス 128
  Reply クラス 152
  Request クラス 145
  Request_Constant クラス 177
  Request_Exp クラス 168
  Request_Exp_Type クラス 174
  Runtime_Data クラス 195
  Runtime_Data_Desc クラス 188
  Runtime_Data_Desc_List クラス 192
  Runtime_Data_List クラス 203
  Server_Info クラス 13
  Unfenced_Generic_Nickname クラス 107
  Unfenced_Generic_Server クラス 79
  Unfenced_Generic_User クラス 97

メンバー関数 *(続き)*
Unfenced_Generic_Wrapper クラス 65
User_Info クラス 23
Wrapper_Info クラス 4
Wrapper_Utilities クラス 205

[ヤ行]

ユーザー・クラス
リスト 97
Fenced_Generic_User 103
Unfenced_Generic_User 97
要求クラス
リスト 144
Predicate_List 181
Reply 152
Request 145
Request_Constant 177
Request_Exp 168
Request_Exp_Type 174

[ラ行]

ラッパー・クラス
リスト 64
Fenced_Generic_Wrapper クラス 72
Unfenced_Generic_Wrapper 65

C

Catalog_Option クラス 2
Column_Info クラス 30
C++ API
カタログ・クラス
リスト 1
Catalog_Option 2
Column_Info 30
Nickname_Info 49
Server_Info 13
User_Info 23
Wrapper_Info 4
サーバー・クラス
リスト 78
Fenced_Generic_Server 90
C++ API *(続き)*
サーバー・クラス *(続き)*
Unfenced_Generic_Server 79
操作クラス
リスト 128
Remote_Passthru 138
Remote_Query 128
データ・クラス
リスト 187
Runtime_Data 195
Runtime_Data_Desc 188
Runtime_Data_Desc_List 192
Runtime_Data_List 203
ニックネーム・クラス
リスト 106
Fenced_Generic_Nickname 115
Unfenced_Generic_Nickname 107
ユーザー・クラス
リスト 97
Fenced_Generic_User 103
Unfenced_Generic_User 97
要求クラス
リスト 144
Predicate_List 181
Reply 152
Request 145
Request_Constant 177
Request_Exp 168
Request_Exp_Type 174
ラッパー・クラス
リスト 64
Fenced_Generic_Wrapper 72
Unfenced_Generic_Wrapper 65
C++ クラスおよびメソッド 1

F

Fenced_Generic_Nickname クラス 115
Fenced_Generic_Server クラス 90
Fenced_Generic_User クラス 103
Fenced_Generic_Wrapper クラス 72

N

Nickname_Info クラス 49

P

Predicate_List クラス 181

R

Remote_Connection クラス 121
Remote_Passthru クラス 138
Remote_Query クラス 128
Reply クラス 152
Request クラス 145
Request_Constant クラス 177
Request_Exp クラス 168
Request_Exp_Type クラス 174
Runtime_Data クラス 195
Runtime_Data_Desc クラス 188
Runtime_Data_Desc_List クラス 192
Runtime_Data_List クラス 203

S

Server_Info クラス 13

U

Unfenced_Generic_Nickname クラス 107
Unfenced_Generic_Server クラス 79
Unfenced_Generic_User クラス 97
Unfenced_Generic_Wrapper クラス 65
User_Info クラス 23

W

Wrapper_Info クラス 4
Wrapper_Utilities クラス 205

IBM と連絡を取る

お住まいの国または地域の IBM 事業所を探すには、 www.ibm.com/planetwide の IBM Directory of Worldwide Contacts をお調べください。

製品情報

DB2 Information Integrator に関する情報は、電話または Web で入手可能です。

Web をご利用の場合は、www.ibm.com/software/data/integration にアクセスしてください。このサイトには、技術ライブラリー、資料の注文方法、クライアント・ダウンロード、ニュースグループ、フィックスパック、ニュース、および Web リソースへのリンクに関する最新情報が掲載されています。

お住まいの国または地域の IBM 事業所を探すには、 www.ibm.com/planetwide の IBM Directory of Worldwide Contacts をお調べください。

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

IBM DB2 Information Integrator

ラッパー開発者向け API リファレンス

バージョン 8