IBM DB2 Information Integrator

Java API Reference for Developing Wrappers

Version 8.2

SC18-9173-00

IBM DB2 Information Integrator

Java API Reference for Developing Wrappers

Version 8.2

SC18-9173-00

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 209.

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

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

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

Contents

About this book

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

Who should read this book

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

Java classes and methods

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

- Catalog classes
- Wrapper classes
- Server classes
- User classes
- Nickname classes
- A connection class
- Operation classes
- Planning classes
- Data classes
- Exception and utilities classes

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

Catalog classes for the Java API

The following table describes each catalog class for the Java API.

Table 1. Catalog classes

Class name	Description
CatalogInfo	The class that represents the base class for all the catalog classes and provides the infrastructure to manage a list of options.
ColumnInfo	The class that encapsulates catalog information for a column of a nickname. This class includes column-statistical information.
NicknameInfo	The class that encapsulates catalog information for a nickname, including column definitions, from the CREATE NICKNAME and ALTER NICKNAME statements.
ServerInfo	The class that encapsulates the catalog information for a server object from the CREATE SERVER and ALTER SERVER statements.
UserInfo	The class that encapsulates the catalog information for a user mapping from the CREATE USER MAPPING and ALTER USER MAPPING statements.
WrapperInfo	The class that encapsulates the catalog information for a wrapper object from the CREATE WRAPPER and ALTER WRAPPER statements.
CatalogOption	The class that represents the base class for options of the catalog objects. This class encapsulates one or more name-value pairs and links to the next option and the previous option.

Related reference:

- "CatalogInfo class (Java)" on page 2
- "CatalogOption class (Java)" on page 53
- "ColumnInfo class (Java)" on page 5
- "NicknameInfo class (Java)" on page 24
- "ServerInfo class (Java)" on page 34
- "UserInfo class (Java)" on page 44
- "WrapperInfo class (Java)" on page 48

CatalogInfo class (Java)

This topic describes the CatalogInfo class and provides details for the methods.

The CatalogInfo class does not contain any constructors.

Overview

The CatalogInfo class represents the base class for all the catalog classes and provides the infrastructure to manage a list of options.

This class is a collection style class that stores a sequential list of options that can be accessed either by their sequences or by their option names. This class uses the current option and the methods that relate to the option.

The ColumnInfo, NicknameInfo, ServerInfo, UserInfo, and WrapperInfo classes are subclasses of the CatalogInfo class.

The CatalogInfo class is one of the catalog classes for the Java API.

Package

com.ibm.db2.wrapper

Methods

The following table describes the methods of the CatalogInfo class. The methods are described in more detail after the table.

Table 2. Methods for the CatalogInfo class

Method	Description
addOption	Add an option to the options chain.
dropOption	Delete an option from the options chain.
getFirstOption	Retrieve the first option from the chain.
getNextOption	Retrieve the next option from the chain.
getOption	Retrieve the option object for the specified option name.

addOption method

Purpose

Add an option to the options chain.

Syntax

```
public void addOption(java.lang.String optionName,
                      java.lang.String optionValue,
                      int action,
                      java.lang.String optionType,
                      java.lang.String objectName,
                  throws WrapperException
```

Parameters

Table 3. Parameters for the addOption method

Name	Description
optionName	Name of the option.
optionValue	Value of the option.
action	Action flag for the option. Valid actions for the options are specified in the CatalogOption class.
optionType	Type of option. This parameter value is a token that is used in the SQL1884 error message to identify the option type if this is a duplicate option.
objectName	Name of the object that owns the option. This parameter value is a token that is used in the SQL1884 error message to identify the owner object name if this is a duplicate option.

Return value

None.

Throws

A WrapperException object if the option already exists in the chain, or if the action is invalid.

dropOption method

Purpose

Delete an option from the options chain.

Syntax

```
public void dropOption(CatalogOption option)
                throws WrapperException
```

Parameters

Table 4. Parameters for the dropOption method

Name	Description
option	Option to be deleted.

Return value

None.

Throws

A WrapperException object if the option is null.

getFirstOption method

Purpose

Retrieve the first option from the chain.

Syntax

```
public final CatalogOption getFirstOption()
```

Parameters

None.

Return value

A CatalogOption instance that represents the first option, or null if no options are specified.

Throws

None.

getNextOption method

Purpose

Retrieve the next option from the chain.

Syntax

```
public final CatalogOption getNextOption(CatalogOption currentOption)
```

Parameters

Table 5. Parameters for the getNextOption method

Name	Description
currentOption	Current option in the chain.

Return value

A CatalogOption instance that represents the next option, or null if there are no more options.

Throws

None.

getOption method

Purpose

Retrieve the option object for the specified option name.

Syntax

```
public final CatalogOption getOption(java.lang.String optionName)
                                throws WrapperException
```

Parameters

Table 6. Parameters for the getOption method

Name	Description
optionName	Name of the option.

Return value

A CatalogOption instance that represents the searched option, or null if no option is found.

Throws

A WrapperException object if the option name is null.

Related reference:

- "CatalogOption class (Java)" on page 53
- "WrapperException class (Java)" on page 190

ColumnInfo class (Java)

This topic describes the ColumnInfo class and provides details for the constructors and methods.

Overview

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

The public ColumnInfo class extends the CatalogInfo class.

The ColumnInfo class is one of the catalog classes for the Java API.

Usage The ColumnInfo class is instantiated by the federated server to contain information from either a CREATE NICKNAME or an ALTER NICKNAME statement, or from the federated server's system catalog. This class is instantiated by the wrapper when information is added during CREATE NICKNAME or ALTER NICKNAME statement operations.

Package
com.ibm.db2.wrapper

Constructors and methods

The following tables describe the constructor and the methods of the ColumnInfo class. The constructors and methods are described in more detail after the tables.

Table 7. Constructors for the ColumnInfo class

Constructor	Description
ColumnInfo	Construct a default (empty) column information object.

Table 8. Methods for the ColumnInfo class

Method	Description
addOption	Add an option to the options chain.
getAvgLength	Retrieve the average length of the column.
getCodepage1	Retrieve the single-byte character set (SBCS) code page for the column.
getCodepage2	Retrieve the double-byte character set (DBCS) code page for the column.
getColCard	Retrieve the cardinality of the column.
getColumnID	Retrieve the ID (position) of the column.
getColumnName	Retrieve the name of the column.
getColumnType	Retrieve the type of the column.
getDefault	Retrieve the default value for the column.
getForBitData	Retrieve the FOR BIT DATA flag for the column.

Table 8. Methods for the ColumnInfo class (continued)

Method	Description
getHigh2Key	Retrieve the second-highest value for the column.
getLow2Key	Retrieve the second-lowest value for the column.
getNewColumnName	Retrieve the new column name that is specified in an ALTER COLUMN statement that includes an ALTER COLUMN or SET COLUMN clause to rename the column.
getNextColumn	Retrieve the next column in the columns chain.
getNulls	Retrieve the nulls-allowed flag.
getOrgLength	Retrieve the maximum length (in bytes) for the column.
getOrgScale	Retrieve the numeric scale of the column.
getTypeName	Retrieve the name of the local column type.
getTypeSchema	Retrieve the schema of the local column type.
isAvgLengthValid	Verify whether an average length is specified for the column.
isCodepage1Valid	Verify whether a single-byte character set (SBCS) code page is specified for the column.
isCodepage2Valid	Verify whether a double-byte character set (DBCS) code page is specified for the column.
isColCardValid	Verify whether a cardinality value is specified for the column.
isColumnIDValid	Verify whether a column ID (position) is specified for the column.
isColumnNameValid	Verify whether a name is specified for the column.
isColumnTypeValid	Verify whether a local type is specified for the column.
isDefaultValid	Verify whether a default value is specified for the column.
isForBitDataValid	Verify whether a FOR BIT DATA flag is specified for the column.
isHigh2KeyValid	Verify whether a second-highest value is specified for the column.
isLow2KeyValid	Verify whether a second-lowest value is specified for the column.
isNewColumnNameValid	Verify whether a new name is specified for the column.
isNullsValid	Verify whether a nulls-allowed flag is specified for the column.
isOrgLengthValid	Verify whether an original length is specified for the column.
isOrgScaleValid	Verify whether an original scale is specified for the column.

Table 8. Methods for the ColumnInfo class (continued)

Method	Description
isTypeNameValid	Verify whether a local type name is specified for the column.
isTypeSchemaValid	Verify whether a local type schema is specified for the column.
setAvgLength	Set the average length of the column.
setCodepage1	Set the single-byte character set (SBCS) code page for the column.
setCodepage2	Set tthe double-byte character set (DBCS) code page for the column.
setColCard	Set the cardinality of the column.
setColumnID	Set the column ID (position).
setColumnName	Set the name of the column.
setColumnType	Set the type of the column.
setDefault	Set the default value for the column.
setForBitData	Set the FOR BIT DATA flag for the column.
setHigh2Key	Set the second-highest value for the column.
setLow2Key	Set the second-lowest value for the column.
setNewColumnName	Set the new name for the column.
setNulls	Set the nulls-allowed flag.
setOrgLength	Set the maximum length (in bytes) for the column.
setOrgScale	Set the numeric scale of the column.
setTypeName	Set the name of the local column type.
setTypeSchema	Set the schema of the local column type.

ColumnInfo constructor

Purpose

Construct a default (empty) column information object.

Syntax

```
public ColumnInfo()
```

Parameters

None.

AddOption method

Purpose

Add an option to the options chain.

Syntax

```
public void addOption(java.lang.String optionName,
                      java.lang.String optionValue,
                      int action)
     throws WrapperException
```

Parameters

Table 9. Parameters for the AddOption method

Name	Description
optionName	The name of the option.
optionValue	The value of the option.
action	The action flag for the option. Valid actions for the options are specified in the CatalogOption class.

Return value

None.

Throws

A WrapperException object if the option already exists in the chain, or if the action is invalid.

getAvgLength method

Purpose

Retrieve the average length of the column.

Syntax

```
public int getAvgLength()
```

Parameters

None.

Return value

The average column length.

Throws

None.

getCodepage1 method

Purpose

Retrieve the single-byte character set (SBCS) code page for the column.

Syntax

```
public short getCodepage1()
```

Parameters

None.

Return value

The single-byte character set (SBCS) code page.

Throws

None.

getCodepage2 method

Purpose

Retrieve the double-byte character set (SBCS) code page for the column.

Syntax

```
public short getCodepage2()
```

Parameters

None.

Return value
The single-byte character set (SBCS) code page.

Throws
None.

getColCard method

Purpose
Retrieve the cardinality of the column.

Syntax

```
public long getColCard()
```

Parameters
None.

Return value
The column cardinality.

Throws
None.

getColumnID method

Purpose
Retrieve the column ID. The column ID represents the position of the column. Column position values start at 0.

Syntax

```
public short getColumnID()
```

Parameters
None.

Return value
The ID (position) of the column.

Throws
None.

getColumnName method

Purpose
Retrieve the name of the column.

Syntax

```
public java.lang.String getColumnName()
```

Parameters
None.

Return value
The column name.

Throws
None.

getColumnType method

Purpose
Retrieve the type of the column.

Syntax

```
public java.lang.String getColumnType()
```

Parameters
None.

Return value
The column type.

Throws
None.

getDefault method

Purpose
Retrieve the default value of the column.

Syntax

```
public java.lang.String getDefault()
```

Parameters
None.

Return value
The default value.

Throws
None.

getForBitData method

Purpose
Retrieve the FOR BIT DATA flag for the column.

Syntax

```
public boolean getForBitData()
```

Parameters
None.

Return value
The value that indicates whether the column does or does not have the FOR BIT DATA flag set.

Throws
None.

getHigh2Key method

Purpose
Retrieve the second-highest value for the column.

Syntax

```
public java.lang.String getHigh2Key()
```

Parameters
None.

Return value
The high2Key value.

Throws
None.

getLow2Key method

Purpose
Retrieve the second-lowest value for the column.

Syntax

```
public java.lang.String getLow2Key()
```

Parameters

None.

Return value

The low2Key value.

Throws

None.

getNewColumnName method

Purpose

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

Syntax

```
public java.lang.String getNewColumnName()
```

Parameters

None.

Return value

The new column name.

Throws

None.

getNextColumn method

Purpose

Retrieve the next column in the columns chain.

Syntax

```
public ColumnInfo getNextColumn()
```

Parameters

None.

Return value

The next column in the chain.

Throws

None.

getNulls method

Purpose

Retrieve the nulls-allowed flag.

Syntax

```
public boolean getNulls()
```

Parameters

None.

Return value

The nulls-allowed flag.

Throws

None.

getOrgLength method

Purpose

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

Syntax

```
public int getOrgLength()
```

Parameters

None.

Return value

The column length.

Throws

None.

getOrgScale method

Purpose

Retrieve the numeric scale of the column.

Syntax

```
public short getOrgScale()
```

Parameters

None.

Return value

The column scale.

Throws

None.

getTypeName method

Purpose

Retrieve the name of the local column type.

Syntax

```
public java.lang.String getTypeName()
```

Parameters

None.

Return value

The name of the type.

Throws

None.

getTypeSchema method

Purpose

Retrieve the schema of the local column type.

Syntax

```
public java.lang.String getTypeSchema()
```

Parameters

None.

Return value

The schema name of the type.

Throws
None.

isAvgLengthValid method

Purpose
Verify whether an average length is specified for the column.

Syntax

```
public boolean isAvgLengthValid()
```

Parameters
None.

Return value
A value of true if an average length value is specified. Otherwise, the return value is false.

Throws
None.

isCodepage1Valid method

Purpose
Verify whether a single-byte character set (SBCS) code page is specified for the column.

Syntax

```
public boolean isCodepage1Valid()
```

Parameters
None.

Return value
A value of true if a single-byte character set value is specified. Otherwise, the return value is false.

Throws
None.

isCodepage2Valid method

Purpose
Verify whether a double-byte character set (DBCS) code page is specified for the column.

Syntax

```
public boolean isCodepage2Valid()
```

Parameters
None.

Return value
A value of true if a double-byte character set value is specified. Otherwise, the return value is false.

Throws
None.

isColCardValid method

Purpose
Verify whether a cardinality value is specified for the column.

Syntax

```
public boolean isColCardValid()
```

Parameters
None.

Return value
A value of true if a cardinality value is specified. Otherwise, the return value is false.

Throws
None.

isColumnIDValid method

Purpose
Verify whether a column ID (position) is specified for the column.

Syntax

```
public boolean isColumnIDValid()
```

Parameters
None.

Return value
A value of true if a column ID (position) value is specified. Otherwise, the return value is false.

Throws
None.

isColumnNameValid method

Purpose
Verify whether a name is specified for the column.

Syntax

```
public boolean isColumnNameValid()
```

Parameters
None.

Return value
A value of true if a column name value is specified. Otherwise, the return value is false.

Throws
None.

isColumnTypeValid method

Purpose
Verify whether a local type is specified for the column.

Syntax

```
public boolean isColumnTypeValid()
```

Parameters
None.

Return value
A value of true if a local type value is specified. Otherwise, the return value is false.

Throws
None.

isDefaultValid method

Purpose

Verify whether a default value is specified for the column.

Syntax

```
public boolean isDefaultValid()
```

Parameters

None.

Return value

A value of true if a default value is specified. Otherwise, the return value is false.

Throws

None.

isForBitDataValid method

Purpose

Verify whether a FOR BIT DATA flag is specified for the column.

Syntax

```
public boolean isForBitDataValid()
```

Parameters

None.

Return value

A value of true if a FOR BIT DATA flag value is specified. Otherwise, the return value is false.

Throws

None.

isHigh2KeyValid method

Purpose

Verify whether a second-highest value is specified for the column.

Syntax

```
public boolean isHigh2KeyValid()
```

Parameters

None.

Return value

A value of true if a second-highest value is specified. Otherwise, the return value is false.

Throws

None.

isLow2KeyValid method

Purpose

Verify whether a second-lowest value is specified for the column.

Syntax

```
public boolean isLow2KeyValid()
```

Parameters

None.

Return value
A value of true if a second-lowest value is specified. Otherwise, the return value is false.

Throws
None.

isNewColumnNameValid method

Purpose
Verify whether a new name is specified for the column.

Syntax

```
public boolean isNewColumnNameValid()
```

Parameters
None.

Return value
A value of true if a new name value is specified. Otherwise, the return value is false.

Throws
None.

isNullsValid method

Purpose
Verify whether a nulls-allowed flag is specified for the column.

Syntax

```
public boolean isNullsValid()
```

Parameters
None.

Return value
A value of true if a nulls-allowed flag value is specified. Otherwise, the return value is false.

Throws
None.

isOrgLengthValid method

Purpose
Verify whether an original length is specified for the column.

Syntax

```
public boolean isOrgLengthValid()
```

Parameters
None.

Return value
A value of true if an original length value is specified. Otherwise, the return value is false.

Throws
None.

isOrgScaleValid method

Purpose
Verify whether an original scale is specified for the column.

Syntax

```
public boolean isOrgScaleValid()
```

Parameters

None.

Return value

A value of true if an original scale value is specified. Otherwise, the return value is false.

Throws

None.

isTypeNameValid method

Purpose

Verify whether a local type name is specified for the column.

Syntax

```
public boolean isTypeNameValid()
```

Parameters

None.

Return value

A value of true if a local type name value is specified. Otherwise, the return value is false.

Throws

None.

isTypeSchemaValid method

Purpose

Verify whether a local type schema is specified for the column.

Syntax

```
public boolean isTypeSchemaValid()
```

Parameters

None.

Return value

A value of true if a local type schema value is specified. Otherwise, the return value is false.

Throws

None.

setAvgLength method

Purpose

Set the average length of the column.

Syntax

```
public void setAvgLength(int avgLength)
```

Parameters

Table 10. Parameters for the setAvgLength method

Name	Description
avgLength	Length to be set.

Return value
None.

Throws
None.

setCodepage1 method

Purpose
Set the single-byte character set (SBCS) code page for the column.

Syntax

```
public void setCodepage1(short codepage1)
```

Parameters

Table 11. Parameters for the setCodepage1 method

Name	Description
codepage1	The single-byte character set (SBCS) code page value to be set.

Return value
None.

Throws
None.

setCodepage2 method

Purpose
Set the double-byte character set (DBCS) code page for the column.

Syntax

```
public void setCodepage2(short codepage2)
```

Parameters

Table 12. Parameters for the setCodepage1 method

Name	Description
codepage2	The double-byte character set (DBCS) code page value to be set.

Return value
None.

Throws
None.

setColCard method

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

Syntax

```
public void setColCard(long colCard)
```

Parameters

Table 13. Parameters for the setColCard method

Name	Description
colCard	Value to be set.

Return value
None.

Throws
None.

setColumnID method

Purpose
Set the column ID. The column ID represents the position of the column. Column position values start at 0

Syntax

```
public void setColumnID(short columnID)
```

Parameters

Table 14. Parameters for the setColumnID method

Name	Description
columnID	ID (position) of the column.

Return value
None.

Throws
None.

setColumnName method

Purpose
Set the name of the column.

Syntax

```
public void setColumnName(java.lang.String columnName)
```

Parameters

Table 15. Parameters for the setColumnName method

Name	Description
columnName	Name to be set.

Return value
None.

Throws
None.

setColumnType method

Purpose
Set the type of the column.

Syntax

```
public void setColumnType(java.lang.String columnType)
```

Parameters

Table 16. Parameters for the setColumnType method

Name	Description
columnType	Column type.

Return value
None.

Throws
None.

setDefault method

Purpose
Set the default value for the column.

Syntax

```
public void setDefault(java.lang.String defaultValue)
```

Parameters

Table 17. Parameters for the setDefault method

Name	Description
defaultValue	Default value.

Return value
None.

Throws
None.

setForBitData method

Purpose
Set the FOR BIT DATA flag for the column.

Syntax

```
public void setForBitData(boolean forBitData)
```

Parameters

Table 18. Parameters for the setForBitData method

Name	Description
forBitData	FOR BIT DATA flag.

Return value
None.

Throws
None.

setHigh2Key method

Purpose
Set the second-highest value for the column. The wrapper can set the second-highest value of a column during CREATE NICKNAME or ALTER

NICKNAME statement processing. The DB2 Universal Database optimizer can use this second-highest value or the highest value when the optimizer develops a query optimization plan.

Syntax

```
public void setHigh2Key(java.lang.String high2Key)
```

Parameters

Table 19. Parameters for the setHigh2Key method

Name	Description
high2Key	Value to be set.

Return value

None.

Throws

None.

setLow2Key method

Purpose

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

Syntax

```
public void setLow2Key(java.lang.String low2Key)
```

Parameters

Table 20. Parameters for the setLow2Key method

Name	Description
low2Key	Value to be set.

Return value

None.

Throws

None.

setNewColumnName method

Purpose

Set the new name for the column. Set the new column name that is specified in an ALTER COLUMN statement that includes an ALTER or SET COLUMN clause to rename the column.

Syntax

```
public void setNewColumnName(java.lang.String newColumnName)
```

Parameters

Table 21. Parameters for the setNewColumnName method

Name	Description
newColumnName	New column name to be set.

Return value
None.

Throws
None.

setNulls method

Purpose
Set the nulls-allowed flag.

Syntax

```
public void setNulls(boolean nulls)
```

Parameters

Table 22. Parameters for the setNulls method

Name	Description
nulls	True to allow null values. False to disallow null values.

Return value
None.

Throws
None.

setOrgLength method

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

Syntax

```
public void setOrgLength(int orgLength)
```

Parameters

Table 23. Parameters for the setOrgLength method

Name	Description
orgLength	Length to be set.

Return value
None.

Throws
None.

setOrgScale method

Purpose
Set the numeric scale of the column.

Syntax

```
public void setOrgScale(short orgScale)
```

Parameters

Table 24. Parameters for the setOrgScale method

Name	Description
orgScale	Scale to be set.

Return value
None.

Throws
None.

setTypeName method

Purpose
Set the name of the local column type.

Syntax

```
public void setTypeName(java.lang.String typeName)
```

Parameters

Table 25. Parameters for the setTypeName method

Name	Description
typeName	Name of the type.

Return value
None.

Throws
None.

setTypeSchema method

Purpose
Set the schema of the local column type.

Syntax

```
public void setTypeSchema(java.lang.String typeSchema)
```

Parameters

Table 26. Parameters for the setTypeSchema method

Name	Description
typeSchema	Schema name of the type.

Return value
None.

Throws
None.

Inherited methods

The following methods are inherited from the CatalogInfo class and can be used with the ColumnInfo class:

- addOption
- dropOption
- getFirstOption
- getNextOption
- getOption

Related reference:

- "CatalogInfo class (Java)" on page 2

• "CatalogOption class (Java)" on page 53

NicknameInfo class (Java)

This topic describes the NicknameInfo class and provides details for the constructors and methods.

Overview

The class that encapsulates catalog information for a nickname, including column definitions, from the CREATE NICKNAME and ALTER NICKNAME statements.

The NicknameInfo class extends the CatalogInfo class.

The NicknameInfo class is one of the catalog classes for the Java API.

Usage The NicknameInfo is instantiated by the federated server to contain information from a CREATE NICKNAME or an ALTER NICKNAME statement or to contain information from the federated server's system catalog. This class is instantiated by the wrapper when information is added during CREATE NICKNAME or ALTER NICKNAME statement operations.

Package
com.ibm.db2.wrapper

Constructors and methods

The following table describes the constructors and methods of the NicknameInfo class. The constructor and methods are described in more detail after the tables.

Table 27. Constructors for the NicknameInfo class

Constructor	Description
NicknameInfo	Construct a default (empty) nickname information object.

Table 28. Methods for the NicknameInfo class

Method	Description
addOption	Add an option to the options chain.
getCard	Retrieve the cardinality value.
getColumn	Retrieve the column with the specified name.
getColumnWithRemoteColumnName	Retrieve the column with the specified remote column name.
getFirstColumn	Retrieve the first column information object.
getFPages	Retrieve the fpages statistics for the nickname.
getNextColumn	Retrieve the next column information object.
getNickname	Retrieve the name of the nickname.
getNPages	Retrieve the npages statistics for the nickname.
getNumColumns	Retrieve the number of columns.
getOverflow	Retrieve the overflow statistics for the nickname.

Table 28. Methods for the NicknameInfo class (continued)

Method	Description
getSchema	Retrieve the local schema name of the nickname.
getServerName	Retrieve the data source server name of the nickname.
insertColumn	Insert a column information object at the position given by its column ID field.
isCardValid	Verify whether a cardinality value is specified.
isFPagesValid	Verify whether the fpages statistic is specified.
isNicknameValid	Verify whether the name of the nickname is specified.
isNPagesValid	Verify whether the npages statistic is specified.
isOverflowValid	Verify whether an overflow statistic is specified.
isSchemaValid	Verify whether a schema name is specified.
isServerNameValid	Verify whether a data source server name is specified.
setCard	Set the cardinality value.
setFPages	Set the fpages statistics for the nickname.
setNickname	Set the name of the nickname.
setNPages	Set the npages statistics for the nickname.
setOverflow	Set the overflow statistics for the nickname.
setSchema	Set the local schema name of the nickname.
setServerName	Set the data source server name of the nickname.

NicknameInfo constructor

Purpose

Construct a default (empty) nickname information object.

Syntax

```
public NicknameInfo()
```

Parameters

None.

addOption method

Purpose

Add an option to the options chain.

Syntax

```
public void addOption(java.lang.String optionName,
                      java.lang.String optionValue,
                      int action)
                throws WrapperException
```

Parameters

Table 29. Parameters for the addOption method

Name	Description
optionName	Name of the option.
optionValue	Value of the option.
action	Action flag for the option. Valid actions for the options are specified in the CatalogOption class.

Return value

None.

Throws

A WrapperException object if the option already exists in the chain, or if the action is invalid.

getCard method

Purpose

Retrieve the cardinality value.

Syntax

```
public long getCard()
```

Parameters

None.

Return value

The cardinality value.

Throws

None.

getColumn method

Purpose

Retrieve the column with the specified name.

Syntax

```
public ColumnInfo getColumn(java.lang.String name)
```

Parameters

Table 30. Parameters for the getColumn method

Name	Description
name	Name of the retrieved column.

Return value

The column descriptor.

Throws

None.

getColumnWithRemoteColumnName method

Purpose

Retrieve the column with the specified remote column name.

Syntax

```
public ColumnInfo getColumnWithRemoteColumnName(java.lang.String remoteColumnName)
                                                throws WrapperException
```

Parameters

Table 31. Parameters for the getColumnWithRemoteColumnName method

Name	Description
remoteColumnName	Remote name of the retrieved column.

Return value
The column descriptor.

Throws
A WrapperException object if the method fails.

getFirstColumn method

Purpose
Retrieve the first column information object.

Syntax

```
public ColumnInfo getFirstColumn()
```

Parameters
None.

Return value
The first column descriptor.

Throws
None.

getFPages method

Purpose
Retrieve the fpages statistics for the nickname.

Syntax

```
public int getFPages()
```

Parameters
None.

Return value
The fpages statistics value.

Throws
None.

getNextColumn method

Purpose
Retrieve the next column.

Syntax

```
public ColumnInfo getNextColumn(ColumnInfo currentColumn)
```

Parameters

Table 32. Parameters for the getNextColumn method

Name	Description
currentColumn	Current column descriptor.

Return value
The next column descriptor.

Throws
None.

getNickname method

Purpose
Retrieve the name of the nickname.

Syntax

```
public java.lang.String getNickname()
```

Parameters
None.

Return value
The name.

Throws
None.

getNPages method

Purpose
Retrieve the npages statistics for the nickname.

Syntax

```
public int getNPages()
```

Parameters
None.

Return value
The npages statistics value.

Throws
None.

getNumColumns method

Purpose
Retrieve the number of columns.

Syntax

```
public int getNumColumns()
```

Parameters
None.

Return value
The number of columns.

Throws
None.

getOverflow method

Purpose
Retrieve the overflow statistics for the nickname.

Syntax

```
public int getOverflow()
```

Parameters
None.

Return value
The overflow statistics value.

Throws
None.

getSchema method

Purpose
Retrieve the local schema name of the nickname.

Syntax

```
public java.lang.String getSchema()
```

Parameters
None.

Return value
The local schema name.

Throws
None.

getServerName method

Purpose
Retrieve the data source server name of the nickname.

Syntax

```
public java.lang.String getServerName()
```

Parameters
None.

Return value
The data source server name.

Throws
None.

insertColumn method

Purpose
Insert a column information object at the position given by its column ID field.

Syntax

```
public void insertColumn(ColumnInfo newColumn)
                  throws WrapperException
```

Parameters

Table 33. Parameters for the insertColumn method

Name	Description
newColumn	Column information object to insert.

Return value
None.

Throws
A WrapperException object if the ColumnInfo object is null.

isCardValid method

Purpose

Verify whether a cardinality value is specified.

Syntax

```
public boolean isCardValid()
```

Parameters

None.

Return value

A value of true if a cardinality value is specified. Otherwise, the return value is false.

Throws

None.

isFPagesValid method

Purpose

Verify whether the fpages statistic is specified.

Syntax

```
public boolean isFPagesValid()
```

Parameters

None.

Return value

A value of true if the statistic is specified. Otherwise, the return value is false.

Throws

None.

isNicknameValid method

Purpose

Verify whether the name of the nickname is specified.

Syntax

```
public boolean isNicknameValid()
```

Parameters

None.

Return value

A value of true if a name is specified. Otherwise, the return value is false.

Throws

None.

isNPagesValid method

Purpose

Verify whether the npages statistic is specified.

Syntax

```
public boolean isNPagesValid()
```

Parameters

None.

Return value
A value of true if the statistic is specified. Otherwise, the return value is false.

Throws
None.

isOverflowValid method

Purpose
Verify whether an overflow statistic is specified.

Syntax

```
public boolean isOverflowValid()
```

Parameters
None.

Return value
A value of true if an overflow statistics value is specified. Otherwise, the return value is false.

Throws
None.

isSchemaValid method

Purpose
Verify whether a schema name is specified.

Syntax

```
public boolean isSchemaValid()
```

Parameters
None.

Return value
A value of true if a schema name is specified. Otherwise, the return value is false.

Throws
None.

isServerNameValid method

Purpose
Verify whether a data source server name is specified.

Syntax

```
public boolean isServerNameValid()
```

Parameters
None.

Return value
A value of true if a data source server name is specified. Otherwise, the return value is false.

Throws
None.

setCard method

Purpose

Set the cardinality value.

Syntax

```
public void setCard(long card)
```

Parameters

Table 34. Parameters for the setCard method

Name	Description
card	Cardinality value.

Return value

None.

Throws

None.

setFPages method

Purpose

Set the fpages statistics for the nickname.

Syntax

```
public void setFPages(int fPages)
```

Parameters

Table 35. Parameters for the setFPages method

Name	Description
fPages	An fpages statistics value.

Return value

None.

Throws

None.

setNickname method

Purpose

Set the name of the nickname.

Syntax

```
public void setNickname(java.lang.String nickname)
```

Parameters

Table 36. Parameters for the setNickname method

Name	Description
nickname	Name of the nickname.

Return value
None.

Throws
None.

setNPages method

Purpose
Set the npages statistics for the nickname.

Syntax

```
public void setNPages(int nPages)
```

Parameters

Table 37. Parameters for the setNPages method

Name	Description
nPages	An npages statistics value.

Return value
None.

Throws
None.

setOverflow method

Purpose
Set the overflow statistics for the nickname.

Syntax

```
public void setOverflow(int overflow)
```

Parameters

Table 38. Parameters for the setOverflow method

Name	Description
overflow	An overflow statistics value.

Return value
None.

Throws
None.

setSchema method

Purpose
Set the local schema name of the nickname.

Syntax

```
public void setSchema(java.lang.String schema)
```

Parameters

Table 39. Parameters for the setSchema method

Name	Description
schema	Local schema name.

Return value
None.

Throws
None.

setServerName method

Purpose
Set the data source server name of the nickname.

Syntax

```
public void setServerName(java.lang.String serverName)
```

Parameters

Table 40. Parameters for the setServerName method

Name	Description
serverName	A data source server name.

Return value
None.

Throws
None.

Inherited methods

The following methods are inherited from the CatalogInfo class and can be used with the NicknameInfo class:

- addOption
- dropOption
- getFirstOption
- getNextOption
- getOption

Related reference:

- "CatalogInfo class (Java)" on page 2
- "CatalogOption class (Java)" on page 53
- "ColumnInfo class (Java)" on page 5
- "WrapperException class (Java)" on page 190

ServerInfo class (Java)

This topic describes the ServerInfo class and provides details for the constructors and the methods.

Overview

The ServerInfo class encapsulates the catalog information for a data source server object from CREATE SERVER and ALTER SERVER statements, and extends the CatalogInfo class.

The ServerInfo class is one of the catalog classes for the Java API.

Usage The ServerInfo class is instantiated by the federated server to contain information either from a CREATE SERVER statement or an ALTER SERVER statement, or from the federated server's system catalog. This class is also instantiated by the wrapper when information is added during CREATE SERVER or ALTER SERVER statement operations.

Package
com.ibm.db2.wrapper

Constructors and methods

The following table describes the constructors and methods of the ServerInfo class. The constructors and methods are described in more detail after the tables.

Table 41. Constructors for the ServerInfo class

Constructor	Description
ServerInfo	Construct a default (empty) server information object.
ServerInfo	Construct a fully initialized server information object.

Table 42. Methods for the ServerInfo class

Method	Description
addOption	Add a single-value option to the options chain.
addOption	Add a multi-value option to the options chain.
dropOption	Delete an option from the options chain.
dropOption	Delete a value from a multi-value option. If no values remain, delete the entire option.
getAuthID	Retrieve the authorization ID for the server.
getPassword	Retrieve the password for the server.
getServerName	Retrieve the name of the server.
getType	Retrieve the type of the server.
getVersion	Retrieve the version string.
getWrapperName	Retrieve the name of the wrapper that the data source server belongs to.
isAuthIDValid	Verify whether an authorization ID value is specified.
isNameValid	Verify whether a name value is specified.
isPasswordValid	Verify whether a password value is specified.
isTypeValid	Verify whether a type value is specified.
isVersionValid	Verify whether a version value is specified.

Table 42. Methods for the ServerInfo class (continued)

Method	Description
isWrapperNameValid	Verify whether a wrapper name value is specified.
setAuthID	Set the authorization ID for the data source server.
setPassword	Set the password for the data source server.
setServerName	Set the data source server name.
setType	Set the type of the data source server.
setVersion	Set the version string.
setWrapperName	Set the name of the wrapper name that the data source server belongs to.

ServerInfo constructor

Purpose

Construct a default (empty) server information object.

Syntax

```
public ServerInfo()
```

Parameters

None.

ServerInfo constructor

Purpose

Construct a fully initialized server information object.

Syntax

```
public ServerInfo(java.lang.String name,
                  java.lang.String type,
                  java.lang.String version,
                  java.lang.String wrapper)
```

Parameters

Table 43. Parameters for the ServerInfo constructor

Name	Description
name	Server name.
type	Server type.
version	Version.
wrapper	Wrapper name that the server belongs to.

addOption method

Purpose

Add a single-value option to the options chain.

Syntax

```
public void addOption(java.lang.String optionName,
                      java.lang.String optionValue,
                      int action)
               throws WrapperException
```

Parameters

Table 44. Parameters for the addOption constructor

Name	Description
optionName	Name of the option.
optionValue	Value of the option.
action	Action flag for the option.

Return value

None.

Throws

A WrapperException object if the option already exists in the chain, or if the action is invalid.

addOption method

Purpose

Add a multi-value option to the options chain.

Syntax

```
public void addOption(java.lang.String optionName,
                      java.lang.String optionValue,
                      java.sql.Timestamp timestamp,
                      java.lang.String valueID,
                      int action)
               throws WrapperException
```

Parameters

Table 45. Parameters for the addOption constructor

Name	Description
optionName	Name of the option.
optionValue	Value of the option.
timestamp	Timestamp of the value.
valueID	ID of the value.
action	Action flag for the option.

Return value

None.

Throws

A WrapperException object if a duplicate value ID is specified, or if the action is invalid.

dropOption method

Purpose

Delete an option from the options chain.

Syntax

```
public void dropOption(CatalogOption option)
               throws WrapperException
```

Parameters

Table 46. Parameters for the dropOption method

Name	Description
option	Option to delete.

Return value
None.

Throws
A WrapperException object if the option object is null.

dropOption method

Purpose
Delete a value from a multi-value option. If no values remain, delete the entire option.

Syntax

```
public void dropOption(CatalogOption option,
                       java.lang.String valueID)
                throws WrapperException
```

Parameters

Table 47. Parameters for the dropOption method

Name	Description
option	Option.
valueID	ID of the value to be deleted.

Return value
None.

Throws
A WrapperException object if the option object is null or if the value ID is not found.

getAuthID method

Purpose
Retrieve the authorization ID for the data source server.

Syntax

```
public java.lang.String getAuthID()
```

Parameters
None.

Return value
The authorization ID.

Throws
None.

getPassword method

Purpose
Retrieve the password for the data source server.

Syntax

```
public java.lang.String getPassword()
```

Parameters
None.

Return value
The password.

Throws
None.

getServerName method

Purpose
Retrieve the name of the data source server.

Syntax

```
public java.lang.String getServerName()
```

Parameters
None.

Return value
The server name.

Throws
None.

getType method

Purpose
Retrieve the type of the data source server.

Syntax

```
public java.lang.String getType()
```

Parameters
None.

Return value
The server type.

Throws
None.

getVersion method

Purpose
Retrieve the version string.

Syntax

```
public java.lang.String getVersion()
```

Parameters
None.

Return value
The version.

Throws
None.

getWrapperName method

Purpose
Retrieve the name of the wrapper that the data source server belongs to.

Syntax

```
public java.lang.String getWrapperName()
```

Parameters
None.

Return value
The wrapper name.

Throws
None.

isAuthIDValid method

Purpose
Verify whether an authorization ID value is specified.

Syntax

```
public boolean isAuthIDValid()
```

Parameters
None.

Return value
A value of true if an authorization ID value is specified. Otherwise, the value is false.

Throws
None.

isNameValid method

Purpose
Verify whether a name value is specified.

Syntax

```
public boolean isNameValid()
```

Parameters
None.

Return value
A value of true if a name value is specified. Otherwise, the value is false.

Throws
None.

isPasswordValid method

Purpose
Verify whether a password value is specified.

Syntax

```
public boolean isPasswordValid()
```

Parameters
None.

Return value
A value of true if a password value is specified. Otherwise, the value is false.

Throws
None.

isTypeValid method

Purpose

Verify whether a type value is specified.

Syntax

```
public boolean isTypeValid()
```

Parameters

None.

Return value

A value of true if a type value is specified. Otherwise, the value is false.

Throws

None.

isVersionValid method

Purpose

Verify whether a version value is specified.

Syntax

```
public boolean isVersionValid()
```

Parameters

None.

Return value

A value of true if a version value is specified. Otherwise, the value is false.

Throws

None.

isWrapperNameValid method

Purpose

Verify whether a wrapper name value is specified.

Syntax

```
public boolean isWrapperNameValid()
```

Parameters

None.

Return value

A value of true if a wrapper name value is specified. Otherwise, the value is false.

Throws

None.

setAuthID method

Purpose

Set the authorization ID for the data source server.

Syntax

```
public void setAuthID(java.lang.String authID)
```

Parameters

Table 48. Parameters for the setAuthID method

Name	Description
authID	Authorization ID.

Return value
None.

Throws
None.

setPassword method

Purpose
Set the password for the data source server.

Syntax

```
public void setPassword(java.lang.String password)
```

Parameters

Table 49. Parameters for the setPassword method

Name	Description
password	The password for the data source server.

Return value
None.

Throws
None.

setServerName method

Purpose
Set the data source server name.

Syntax

```
public void setServerName(java.lang.String name)
```

Parameters

Table 50. Parameters for the setServerName method

Name	Description
name	The data source server name.

Return value
None.

Throws
None.

setType method

Purpose
Set the type of the data source server.

Syntax

```
public void setType(java.lang.String type)
```

Parameters

Table 51. Parameters for the setType method

Name	Description
type	The data source erver type.

Return value
None.

Throws
None.

setVersion method

Purpose
Set the version string.

Syntax

```
public void setVersion(java.lang.String version)
```

Parameters

Table 52. Parameters for the setVersion method

Name	Description
version	Version.

Return value
None.

Throws
None.

setWrapperName method

Purpose
Set the name of the wrapper name that the data source server belongs to.

Syntax

```
public void setWrapperName(java.lang.String name)
```

Parameters

Table 53. Parameters for the setWrapperName method

Name	Description
name	Wrapper name.

Return value
None.

Throws
None.

Inherited methods

The following methods are inherited from the CatalogInfo class and can be used with the ServerInfo class:

- addOption
- getFirstOption

- getNextOption
- getOption

Related reference:

- "CatalogInfo class (Java)" on page 2
- "CatalogOption class (Java)" on page 53
- "WrapperException class (Java)" on page 190

UserInfo class (Java)

This topic describes the UserInfo class and provides details for the constructor and the methods.

Overview

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

The public UserInfo class extends the CatalogInfo class.

The UserInfo class is one of the catalog classes for the Java API.

Usage The UserInfo class is instantiated by the federated server to contain information from a CREATE USER MAPPING or an ALTER USER MAPPING statement or to contain information from the federated server's system catalog. This class is instantiated by the wrapper when information is added during CREATE USER MAPPING or ALTER USER MAPPING statement processing.

Package
com.ibm.db2.wrapper

Constructors and methods

The following tables describe the constructor and the methods of the UserInfo class. The constructor and methods are described in more detail after the tables.

Table 54. Constructors for the UserInfo class

Constructor	Description
UserInfo	Construct a default (empty) user information object.

Table 55. Methods for the UserInfo class

Method	Description
addOption	Add an option to the options chain.
getAuthID	Retrieve the authorization ID for this user mapping.
getPassword	Retrieve the password for this user mapping.
getServerName	Retrieve the data source server name for this user mapping.
isAuthIDValid	Verify whether an authorization ID is specified.

Table 55. Methods for the UserInfo class (continued)

Method	Description
isServerNameValid	Verify whether a data source server name is specified.
setAuthID	Set the authorization ID for this user mapping.
setServerName	Set the data source server name for this user mapping.

UserInfo constructor

Purpose

Construct a default (empty) user information object.

Syntax

```
public UserInfo()
```

Parameters

None.

addOption method

Purpose

Add an option to the options chain.

Syntax

```
public void addOption(java.lang.String optionName,
                      java.lang.String optionValue,
                      int action)
               throws WrapperException
```

Parameters

Table 56. Parameters for the addOption method

Name	Description
optionName	Name of the option.
optionValue	Value of the option.
action	Action flag for the option. Valid actions for the options are specified in the CatalogOption class.

Return value

None.

Throws

A WrapperException object if the option already exists in the chain, or if the action is invalid.

getAuthID method

Purpose

Retrieve the authorization ID for this user mapping.

Syntax

```
public java.lang.String getAuthID()
```

Parameters

None.

Return value
The authorization ID.

Throws
None.

getPassword method

Purpose
Retrieve the password for this user mapping. Obtains the value of the REMOTE_PASSWORD option specified for this user mapping, if any.

Syntax

```
public java.lang.String getPassword()
```

Parameters
None.

Return value
The password specified as the value of the REMOTE_PASSWORD option for this user mapping, or null if the option is not found.

Throws
None.

getServerName method

Purpose
Retrieve the data source server name for this user mapping.

Syntax

```
public java.lang.String getServerName()
```

Parameters
None.

Return value
The data source server name.

Throws
None.

isAuthIDValid method

Purpose
Verify whether an authorization ID is specified.

Syntax

```
public boolean isAuthIDValid()
```

Parameters
None.

Return value
A value of true if an authorization ID value is specified. Otherwise, the value is false.

Throws
None.

isServerNameValid method

Purpose
Verify whether a data source server name is specified.

Syntax

```
public boolean isServerNameValid()
```

Parameters

None.

Return value

A value of true if a data source server name value is specified. Otherwise, the value is false.

Throws

None.

setAuthID method

Purpose

Set the authorization ID for this user mapping.

Syntax

```
public void setAuthID(java.lang.String authID)
```

Parameters

Table 57. Parameters for the setAuthID method

Name	Description
authID	Authorization ID.

Return value

None.

Throws

None.

setServerName method

Purpose

Set the data source server name for this user mapping.

Syntax

```
public void setServerName(java.lang.String serverName)
```

Parameters

Table 58. Parameters for the setServerName method

Name	Description
serverName	Data source server name.

Return value

None.

Throws

None.

Inherited methods

The following methods are inherited from the CatalogInfo class and can be used with the UserInfo class:

- addOption
- dropOption
- getFirstOption

- getNextOption
- getOption

Related reference:

- "CatalogInfo class (Java)" on page 2
- "CatalogOption class (Java)" on page 53

WrapperInfo class (Java)

This topic describes the WrapperInfo class and provides details for the constructor and the methods.

Overview

The WrapperInfo class encapsulates the catalog information for a wrapper from the CREATE WRAPPER and ALTER WRAPPER statements.

The public WrapperInfo class extends the CatalogInfo class.

The WrapperInfo class is one of the catalog classes for the Java API.

Usage The WrapperInfo class is instantiated by the federated server to contain information either from a CREATE WRAPPER or an ALTER WRAPPER statement, or to contain information from the federated server's system catalog. This class is instantiated by the wrapper when information is added during CREATE WRAPPER or ALTER WRAPPER operations.

Package
com.ibm.db2.wrapper

Constructors and methods

The following tables describe the constructor and the methods of the WrapperInfo class. The constructor and methods are described in more detail after the tables.

Table 59. Constructors for the WrapperInfo class

Constructor	Description
WrapperInfo	Construct a new default (empty) wrapper information object.

Table 60. Methods for the WrapperInfo class

Method	Description
addOption	Add an option to the options chain.
getCorelib	Retrieve the core library name.
getType	Retrieve the wrapper type.
getVersion	Retrieve the wrapper version.
getWrapperName	Retrieve the wrapper name.
isCorelibValid	Verify whether a core library is specified.
isNameValid	Verify whether a name is specified.
isTypeValid	Verify whether a type is specified.
isVersionValid	Verify whether a version is specified.
setType	Set the wrapper type.

Table 60. Methods for the WrapperInfo class (continued)

Method	Description
setVersion	Set the wrapper version.
setWrapperName	Set the wrapper name.

WrapperInfo constructor

Purpose

Construct a new default (empty) wrapper information object.

Syntax

```
public WrapperInfo()
```

Parameters

None.

addOption method

Purpose

Add an option to the options chain.

Syntax

```
public void addOption(java.lang.String optionName,
                      java.lang.String optionValue,
                      int action)
               throws WrapperException
```

Parameters

Table 61. Parameters for the addOption method

Name	Description
optionName	Name of the option.
optionValue	Value of the option.
action	Action flag for the option. Valid actions for the options are specified in the CatalogOption class.

Return value

None.

Throws

A WrapperException object if the option already exists in the chain, or if the action is invalid.

getCorelib method

Purpose

Retrieve the core library name.

Syntax

```
public java.lang.String getCorelib()
```

Parameters

None.

Return value

The name of the core library.

Throws

None.

getType method

Purpose

Retrieve the wrapper type.

Syntax

```
public char getType()
```

Parameters

None.

Return value

The wrapper type.

Throws

None.

getVersion method

Purpose

Retrieve the wrapper version.

Syntax

```
public int getVersion()
```

Parameters

None.

Return value

The wrapper version.

Throws

None.

getWrapperName method

Purpose

Retrieve the wrapper name.

Syntax

```
public java.lang.String getWrapperName()
```

Parameters

None.

Return value

The name of the wrapper.

Throws

None.

isCorelibValid method

Purpose

Verify whether a core library is specified.

Syntax

```
public boolean isCorelibValid()
```

Parameters

None.

Retrieve value

A value of true if a core library value is specified. Otherwise, the value is false.

Throws
None.

isNameValid method

Purpose
Verify whether a name is specified.

Syntax

```
public boolean isNameValid()
```

Parameters
None.

Return value
A value of true if a name value is specified. Otherwise, the value is false.

Throws
None.

isTypeValid method

Purpose
Verify whether a type is specified.

Syntax

```
public boolean isTypeValid()
```

Parameters
None.

Return value
A value of true if a type value is specified. Otherwise, the value is false.

Throws
None.

isVersionValid method

Purpose
Verify whether a version is specified.

Syntax

```
public boolean isVersionValid()
```

Parameters
None.

Return value
A value of true if a version value is specified. Otherwise, the value is false.

Throws
None.

setType method

Purpose
Set the wrapper type.

Syntax

```
public void setType(char type)
```

Parameters

Table 62. Parameters for the setType method

Name	Description
type	Wrapper type.

Return value
None.

Throws
None.

setVersion method

Purpose
Set the wrapper version.

Syntax

```
public void setVersion(int version)
```

Parameters

Table 63. Parameters for the setVersion method

Name	Description
version	Wrapper version.

Return value
None.

Throws
None.

setWrapperName method

Purpose
Set the wrapper name.

Syntax

```
public void setWrapperName(java.lang.String name)
```

Parameters

Table 64. Parameters for the setWrapperName method

Name	Description
name	Wrapper name.

Return value
None.

Throws
None.

Inherited methods

The following methods are inherited from the Catalog class and can be used with the WrapperInfo class:

- addOption
- dropOption

- getFirstOption
- getNextOption
- getOption

Related reference:

- "CatalogInfo class (Java)" on page 2

CatalogOption class (Java)

This topic describes the CatalogOption class and provides details for the constants and the methods.

The CatalogOption class does not contain any constructors.

The federated server instantiates this class to describe an option from the federated server's system catalog.

Overview

The CatalogOption class represents the base class for options of the catalog objects. This class encapsulates one or more name-value pairs and links to the next option and the previous option.

The CatalogOption class is one of the catalog classes for the Java API.

Package

com.ibm.db2.wrapper

Constants

The following table describes the valid action constants for the options that you can use with the CatalogOption class.

Table 65. Constants for the CatalogOption class

Constant	Definition	Description
ADD	public static final int ADD	Indicates that the option is added to the catalog.
DROP	public static final int DROP	Indicates that the option is dropped from the catalog.
NONE	public static final int NONE	Indicates no action for the option.
SET	public static final int SET	Indicates that the option is set to a new value. The option to set to a new value must already be in the catalog.

Methods

The following table describes the methods of the CatalogOption class. The methods are described in more detail after the tables

Table 66. Methods for the CatalogOption class

Method	Description
getAction	Retrieve the action for the option.

Table 66. Methods for the CatalogOption class (continued)

Method	Description
getName	Retrieve the name of the option.
getNextOption	Retrieve the next option in the chain that this option belongs to.
getPrevOption	Retrieve the previous option in the chain that this option belongs to.
getValue	Retrieve the option value.
isReserved	Indicate whether the federated server reserves the option.

getAction method

Purpose

Retrieve the action for the option.

Syntax

```
public final int getAction()
```

Parameters

None.

Return value

The action.

Throws

None.

getName method

Purpose

Retrieve the name of the option.

Syntax

```
public final java.lang.String getName()
```

Parameters

None.

Return value

The option name.

Throws

None.

getNextOption method

Purpose

Retrieve the next option in the chain that this option belongs to.

Syntax

```
public final CatalogOption getNextOption()
```

Parameters

None.

Return value

The next option in the chain.

Throws

None.

getPrevOption method

Purpose

Retrieve the previous option in the chain that this option belongs to.

Syntax

```
public final CatalogOption getPrevOption()
```

Parameters

None.

Return value

The previous option in the chain.

Throws

None.

getValue method

Purpose

Retrieve the option value.

Syntax

```
public java.lang.String getValue()
```

Parameters

None.

Return value

The option value as a string.

isReserved method

Purpose

Indicate whether the federated server reserves the option.

Syntax

```
public final boolean isReserved()
```

Parameters

None.

Return value

A value of true if the federated server reserves the option. Otherwise, the method returns a value of false.

Throws

None.

Related reference:

• "WrapperException class (Java)" on page 190

Wrapper classes for the Java API

The following table describes each wrapper class for the Java API.

Table 67. Wrapper classes

Class name	Description
Wrapper	The class that represents the base class for a group of data sources. This class provides configuration and initialization of the library, and access to the data source servers that are supported by the wrapper.

Table 67. Wrapper classes (continued)

Class name	Description
FencedWrapper	The class that represents an abstract wrapper on the fenced (untrusted) process space. Do not use this class directly. Instantiating or subclassing the FencedWrapper class directly results in incorrect wrapper behavior. Use the FencedGenericWrapper class.
FencedGenericWrapper	The class that represents a wrapper in the fenced (untrusted) process space.
UnfencedWrapper	The class that represents an abstract wrapper on the unfenced (trusted) process space. Do not use this class directly. Instantiating or subclassing the UnfencedWrapper class directly results in incorrect wrapper behavior. Use the UnfencedGenericWrapper class.
UnfencedGenericWrapper	The class that represents a wrapper on the unfenced (trusted) process space.

Related reference:

- "FencedGenericWrapper class (Java)" on page 60
- "FencedWrapper class (Java)" on page 59
- "UnfencedGenericWrapper class (Java)" on page 64
- "UnfencedWrapper class (Java)" on page 62
- "Wrapper class (Java)" on page 56

Wrapper class (Java)

This topic describes the Wrapper class and provides details for the methods.

The Wrapper class does not contain contructors.

Overview

The Wrapper class represents the base class for a set of data sources. This class provides library initialization services and access to the data source servers that the wrapper supports. The Wrapper class maintains the following information:

- The wrapper name.
- The wrapper core library name. The returned name is the name of the native library that loaded the wrapper.
- A WrapperInfo object that contains all of the information that pertains to this wrapper. This information gets stored in the federated server's system catalog as a result of issuing the DDL statements CREATE WRAPPER or ALTER WRAPPER.

The FencedWrapper class and the UnfencedWrapper class are subclasses of the Wrapper class.

The Wrapper class is a wrapper class for the Java API.

Usage Do not use this class directly, but subclass the FencedGenericWrapper class and the UnfencedGenericWrapper class.

Package

com.ibm.db2.wrapper

Methods

The following table describes the methods of the Wrapper class. The methods are described in more detail after the table.

Table 68. Methods for the Wrapper class

Method	Description
createServer	Instantiate the appropriate subclass of Server for the wrapper.
getCorelib	Retrieve the library name of the wrapper.
getInfo	Retrieve the wrapper information that is stored in the federated server's system catalog as a result of running DDL statements.
getName	Retrieve the name of the wrapper.
getType	Retrieve the wrapper type.
getVersion	Retrieve the version of the wrapper.
initializeMyWrapper	Initialize the wrapper object state from the catalog information object.

createServer method

Purpose

Instantiate the appropriate subclass of Server for the wrapper. The wrapper writer must implement this method to create an instance of the wrapper-specific Server subclass.

Syntax

```
protected Server createServer(java.lang.String serverName)
                        throws java.lang.Exception
```

Parameters

Table 69. Parameters for the createServer method

Name	Description
serverName	Name of the data source server, which is specified on the CREATE SERVER statement.

Return value

The newly created Server instance.

Throws

An Exception object if a new Server instance cannot be created.

getCorelib method

Purpose

Retrieve the library name of the wrapper.

Syntax

```
public final java.lang.String getCorelib()
```

Parameters

None.

Return value

The name of the library that the wrapper is loaded from.

Throws
None.

getInfo method

Purpose
Retrieve the wrapper information that is stored in the federated server's system catalog as a result of running DDL statements.

Syntax

```
public final WrapperInfo getInfo()
```

Parameters
None.

Return value
The WrapperInfo object.

Throws
None.

getName method

Purpose
Retrieve the name of the wrapper.

Syntax

```
public final java.lang.String getName()
```

Parameters
None.

Return value
The name of the wrapper as specified on the CREATE WRAPPER statement.

Throws
None.

getType method

Purpose
Retrieve the wrapper type. The wrapper type must be N if the wrapper supports nonrelational query planning. The getType method returns N. N is the only valid value.

Syntax

```
public char getType()
```

Parameters
None.

Return value
The type of the wrapper.

Throws
None.

getVersion method

Purpose
Retrieve the version of the wrapper, which represents the version that is

currently running. This value can be compared with the version from the time that the wrapper was registered with DB2 Universal Database to assure compatibility.

Syntax

```
public int getVersion()
```

Parameters

None.

Return value

The version of the wrapper.

Throws

None.

initializeMyWrapper method

Purpose

Initialize the wrapper object state from the catalog information object. The default implementation does nothing. If wrapper-specific wrapper options are supported, the wrapper writer can implement this method in the wrapper-specific subclass of UnfencedGenericWrapper and FencedGenericWrapper.

Syntax

```
protected void initializeMyWrapper(WrapperInfo wrapperInfo)
                                throws java.lang.Exception
```

Parameters

Table 70. Parameters for the initializeMyWrapper method

Name	Description
wrapperInfo	WrapperInfo instance that contains the catalog information for the wrapper.

Return value

None.

Throws

An Exception object if the initialization process fails.

Related reference:

- "FencedWrapper class (Java)" on page 59
- "Server class (Java)" on page 67
- "UnfencedWrapper class (Java)" on page 62
- "WrapperInfo class (Java)" on page 48
- "WrapperException class (Java)" on page 190

FencedWrapper class (Java)

This topic describes the FencedWrapper class.

The FencedWrapper class does not contain constructors.

Overview

The FencedWrapper class represents an abstract wrapper on the fenced (untrusted) process space.

The public FencedWrapper class extends the Wrapper class.

The FencedWrapper class is one of the wrapper classes for the Java API.

Usage Do not use this class directly. Instantiating or subclassing the FencedWrapper class directly results in incorrect wrapper behavior. Subclass the FencedGenericWrapper class.

Package
com.ibm.db2.wrapper

Inherited methods

The following methods are inherited from the Wrapper class and can be used with the FencedWrapper class:

- createServer
- getCorelib
- getInfo
- getName
- getType
- getVersion
- initializeMyWrapper

Related reference:

- "Wrapper class (Java)" on page 56
- "WrapperException class (Java)" on page 190

FencedGenericWrapper class (Java)

This topic describes the FencedGenericWrapper class and provides details for the constructor and the method.

Overview

The FencedGenericWrapper class represents a wrapper on the fenced (untrusted) process space.

The public FencedGenericWrapper class extends the FencedWrapper class.

The FencedGenericWrapper class is one of the wrapper classes for the Java API.

Usage The wrapper must implement a subclass of FencedGenericWrapper. The name of the wrapper-specific fenced generic wrapper subclass is specified as the value of the FENCED_WRAPPER_CLASS option in the CREATE WRAPPER statement. This name can also be specified by calling the UnfencedWrapper.setFencedWrapperClass(com.ibm.db2.wrapper. WrapperInfo, java.lang.String) function during the wrapper validation process of the UnfencedWrapper.verifyMyRegisterWrapperInfo(WrapperInfo) function.

Package
com.ibm.db2.wrapper

Constructor and method

The following tables describe the constructor and the method of the FencedGenericWrapper class. The constructor and the method are described in more detail after the tables.

Table 71. Constructors for the FencedGenericWrapper class

Constructor	Description
FencedGenericWrapper	Construct a new FencedGenricWrapper object.

Table 72. Method for the FencedGenericWrapper class

Method	Description
getType	Retrieve the wrapper type.

FencedWrapper constructor

Purpose

Construct a new FencedGenricWrapper object.

Syntax

```
protected FencedGenericWrapper()
```

Parameters

None.

Throws

None.

getType method

Purpose

Retrieve the wrapper type. A default implementation returns a value of N. N is the only valid value. This method overrides the getType method in the Wrapper class.

Syntax

```
public char getType()
```

Parameters

None.

Return value

The wrapper type. By default, this function returns a value of N.

Throws

None.

Inherited methods

The following methods are inherited from the Wrapper class and can be used with the FencedGenericWrapper class:

- createServer
- getCorelib
- getInfo
- getName
- getVersion
- initializeMyWrapper

Related tasks:

- "Wrapper classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "FencedWrapper class (Java)" on page 59
- "Wrapper class (Java)" on page 56
- "WrapperException class (Java)" on page 190

UnfencedWrapper class (Java)

This topic describes the UnfencedWrapper class and provides details for the methods.

The UnfencedWrapper class does not contain constructors.

Overview

The UnfencedWrapper class represents an abstract wrapper on the unfenced (trusted) process space.

The public UnfencedWrapper class extends the Wrapper class.

The UnfencedWrapper class is one of the wrapper classes for the Java API.

Usage Do not use the UnfencedWrapper class directly. Instantiating or subclassing the UnfencedWrapper class directly results in incorrect wrapper behavior. Subclass the UnfencedGenericWrapper class.

Package
com.ibm.db2.wrapper

Methods

The following table describes the methods of the UnfencedWrapper class. The methods are described in more detail after the table.

Table 73. Methods for the UnfencedWrapper class

Method	Description
setFencedWrapperClass	Add the wrapper-specific FencedWrapper subclass name to the WrapperInfo object.
verifyMyAlterWrapperInfo	Validate the information that is specified on an ALTER WRAPPER statement.
verifyMyRegisterWrapperInfo	Validate the information that is specified on a CREATE WRAPPER statement.

setFencedWrapperClass method

Purpose
Add the wrapper-specific FencedWrapper subclass name to the WrapperInfo object.

Usage This method indicates which class needs to be loaded for the fenced part of the wrapper, and passes this information to the federated server.

Syntax

```
public final void setFencedWrapperClass(WrapperInfo wrapperInfo,
                                        java.lang.String className)
                                 throws WrapperException
```

Parameters

Table 74. Parameters for the setFencedWrapperClass method

Name	Description
wrapperInfo	WrapperInfo object where the option is added.
className	FencedWrapper subclass name.

Return value
None.

Throws
A WrapperException object if the method fails.

verifyMyAlterWrapperInfo method

Purpose
Validate the information that is specified on an ALTER WRAPPER statement.

Usage Use this method for verification only. Do not update the internal state of the Wrapper object with the new information that is obtained from the ALTER WRAPPER statement. If the execution of the statement is successful, the federated server ensures that the Wrapper object is destroyed and recreated with the new information. A default implementation is provided that allows only reserved options and does not return more information.

This method must be implemented if wrapper-specific wrapper options are supported. This method can be implemented by the wrapper in the wrapper-specific unfenced wrapper subclass.

Syntax

```
protected WrapperInfo verifyMyAlterWrapperInfo(WrapperInfo wrapperInfo)
                                          throws java.lang.Exception
```

Parameters

Table 75. Parameters for the verifyMyAlterWrapperInfo method

Name	Description
wrapperInfo	WrapperInfo object that contains the information that is provided in the ALTER WRAPPER statement.

Return value
A WrapperInfo object with the information that is added by the wrapper.

Throws
An Exception object if the verification process fails.

verifyMyRegisterWrapperInfo method

Purpose
Validate the information that is specified on a CREATE WRAPPER statement.

Usage A default implementation is provided that allows only reserved options

and does not return more information. This method must be implemented if wrapper-specific wrapper options are supported. This method can be implemented by the wrapper in the wrapper-specific unfenced wrapper subclass.

Syntax

```
protected WrapperInfo verifyMyRegisterWrapperInfo(WrapperInfo wrapperInfo)
                                                  throws java.lang.Exception
```

Parameters

Table 76. Parameters for the verifyMyRegisterWrapperInfo method

Name	Description
wrapperInfo	WrapperInfo object that contains the information that is provided in the CREATE WRAPPER statement.

Return value

A WrapperInfo object with the information that is added by the wrapper.

Throws

An Exception object if the verification process fails.

Inherited methods

The following methods are inherited from the Wrapper class and can be used with the UnfencedWrapper class:

- createServer
- getCorelib
- getInfo
- getName
- getVersion
- initializeMyWrapper

Related reference:

- "WrapperInfo class (Java)" on page 48
- "Wrapper class (Java)" on page 56
- "WrapperException class (Java)" on page 190

UnfencedGenericWrapper class (Java)

This topic describes the UnfencedGenericWrapper class and provides details for the constructor and the method.

Overview

The UnfencedGenericWrapper class represents a wrapper on the unfenced (trusted) process space.

The public UnfencedGenericWrapper class extends the UnfencedWrapper class.

The UnfencedGenericWrapper class is one of the wrapper classes for the Java API.

Usage The wrapper must implement a subclass of UnfencedGenericWrapper. The

name of the user-specific unfenced generic wrapper subclass is specified as the value of the UNFENCED_WRAPPER_CLASS option in the CREATE WRAPPER statement.

Package
com.ibm.db2.wrapper

Constructor and method

The following tables describe the constructor and the method of the UnfencedGenericWrapper class. The constructor and method are described in more detail after the tables.

Table 77. Constructors for the UnfencedGenericWrapper class

Constructor	Description
UnfencedGenericWrapper	Construct a new UnfencedGenericWrapper object.

Table 78. Methods for the UnfencedGenericWrapper class

Method	Description
getType	Retrieve the wrapper type.

UnfencedGenericWrapper constructor

Purpose
Construct a new UnfencedGenericWrapper object.

Syntax

```
protected UnfencedGenericWrapper()
```

Parameters
None.

Throws
None.

getType method

Purpose
Retrieve the wrapper type. A default implementation returns a value of N. N is the only valid value.

Syntax

```
public char getType()
```

Parameters
None.

Return value
The wrapper type. By default, this method returns a value of N.

Throws
None.

Inherited methods

The following methods are inherited from the Wrapper class and can be used with the UnfencedGenericWrapper class:

- createServer

- getCorelib
- getInfo
- getName
- getVersion
- initializeMyWrapper

The following methods are inherited from the UnfencedWrapper class and can be used with the UnfencedGenericWrapper class:

- setFencedWrapperClass
- verifyMyAlterWrapperInfo
- verifyMyRegisterWrapperInfo

Related tasks:

- "Wrapper classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "UnfencedWrapper class (Java)" on page 62
- "Wrapper class (Java)" on page 56
- "WrapperException class (Java)" on page 190

Server classes for the Java API

The following table describes each server class for the Java API.

Table 79. Server classes

Class name	Description
Server	The class that represents the abstract base class for all data source server functionality, and that maps to a specific data source that is supported by the wrapper.
FencedServer	The class that represents a subclass of the Server class and is the abstract base class for all data source server functionality that operates in the fenced (untrusted) process space. Do not use this class directly. Instantiating or subclassing the FencedServer class directly results in incorrect wrapper behavior. Use the FencedGenericServer class.
FencedGenericServer	The class that represents a subclass of the Server class and is the abstract base class for all data source server functionality that operates in the fenced (untrusted) process space.
UnfencedServer	The class that represents a subclass of the Server class and is the abstract base class for all data source server functionality that operates in the unfenced (trusted) process space. Do not use this class directly. Instantiating or subclassing the UnfencedServer class directly results in incorrect wrapper behavior. Use the UnfencedGenericServer class.
UnfencedGenericServer	The class that represents a subclass of the Server class and is the abstract base class for all data source server functionality that operates in the unfenced (trusted) process space.

Related reference:

- "FencedGenericServer class (Java)" on page 73
- "FencedServer class (Java)" on page 71
- "Server class (Java)" on page 67
- "UnfencedGenericServer class (Java)" on page 78
- "UnfencedServer class (Java)" on page 75

Server class (Java)

This topic describes the Server class and provides details for the methods.

This class does not contain constructors.

Overview

The Server class is the abstract base class for all data source server functionality. The Server class maps to a specific data source that is supported by the wrapper and maintains the following information:

- The data source server name.
- A reference to the containing wrapper object.
- A ServerInfo object that contains information for this data source server. This information is stored in the federated server's system catalog after you issue the DDL statements.

The FencedServer class and the UnfencedServer class are subclasses of the Server class.

The Server class is one of the server classes for the Java API.

Usage Do not use this class directly, but subclass the FencedGenericServer class and the UnfencedGenericServer class.

Package

com.ibm.db2.wrapper

Methods

The following table describes the methods of the Server class. The methods are described in more detail after the table.

Table 80. Methods for the Server class

Method	Description
createNickname	Instantiate an appropriate Nickname subclass for this data source server.
createRemoteUser	Instantiate an appropriate RemoteUser subclass for this data source server.
findRemoteUser	Search for a remote user mapping with the local name that is specified in the federated server's system catalog.
getInfo	Retrieve the data source server information that is stored in the federated server's system catalog as a result of running DDL statements.
getName	Retrieve the name of the data source server.

Table 80. Methods for the Server class (continued)

Method	Description
getType	Retrieve the data source server type.
getVersion	Retrieve the version of the data source server.
getWrapper	Retrieve the wrapper object that this data source server belongs to.
initializeMyServer	Initialize the data source server with valid federated server system catalog information.

createNickname method

Purpose

Instantiate an appropriate Nickname subclass for this data source server. The wrapper writer must implement this method in the wrapper-specific data source server subclass.

Syntax

```
protected Nickname createNickname(java.lang.String schemaName,
                                  java.lang.String nickname)
                           throws java.lang.Exception
```

Parameters

Table 81. Parameters for the createNickname method

Name	Description
schemaName	The local schema name of the nickname to be created.
nickname	The local name of the nickname to be created.

Return value

The newly created Nickname instance.

Throws

An Exception object if a new Nickname instance cannot be created.

createRemoteUser method

Purpose

Instantiate an appropriate RemoteUser subclass for this data source server.

Usage The wrapper can implement this method in the wrapper-specific data source server subclass. The wrapper writer must implement this method if a wrapper-specific subclass of the RemoteUser class is implemented.

Syntax

```
protected RemoteUser createRemoteUser(java.lang.String userName)
                                throws java.lang.Exception
```

Parameters

Table 82. Parameters for the createRemoteUser method

Name	Description
userName	Name of the remote user mapping to be created as specified in the CREATE USER MAPPING statement.

Return value
The newly created RemoteUser instance.

Throws
An Exception object if a new RemoteUser instance cannot be created.

findRemoteUser method

Purpose
Search for a remote user mapping with the local name that is specified in the federated server's system catalog. If no remote user mapping with the specified name is found, returns null.

Syntax

```
public final RemoteUser findRemoteUser(java.lang.String userName)
```

Parameters

Table 83. Parameters for the findRemoteUser method

Name	Description
userName	Name of the remote user mapping.

Return value
The RemoteUser instance with the specified name, or null if the remote user mapping is not found.

Throws
None.

getInfo method

Purpose
Retrieve the data source server information that is stored in the federated server's system catalog as a result of running DDL statements.

Syntax

```
public final ServerInfo getInfo()
```

Parameters
None.

Return value
The instance that contains the data source server information.

Throws
None.

getName method

Purpose
Retrieve the name of the data source server.

Syntax

```
public final java.lang.String getName()
```

Parameters
None.

Return value
The name of the data source server that is specified in CREATE SERVER statement.

Throws
None.

getType method

Purpose
Retrieve the data source server type.

Syntax

```
public java.lang.String getType()
```

Parameters
None.

Return value
The type of the data source server.

Throws
None.

getVersion method

Purpose
Retrieve the version of the data source server.

Syntax

```
public java.lang.String getVersion()
```

Parameters
None.

Return value
The version of the data source server.

Throws
None.

getWrapper method

Purpose
Retrieve the wrapper object that this data source server belongs to.

Syntax

```
public final Wrapper getWrapper()
```

Parameters
None.

Return value
The wrapper object.

Throws
None.

initializeMyServer method

Purpose
Initialize the data source server with valid federated server system catalog information.

Usage The wrapper can implement this method in the wrapper-specific data source server subclass. The wrapper writer must implement this method if wrapper-specific data source server options are supported.

Syntax

```
protected void initializeMyServer(ServerInfo serverInfo)
                                  throws java.lang.Exception
```

Parameters

Table 84. Parameters for the initializeMyServer method

Name	Description
serverInfo	ServerInfo instance that contains the data source server information.

Return value
None.

Throws
An Exception object if the initialization process fails.

Related reference:
- "FencedServer class (Java)" on page 71
- "Nickname class (Java)" on page 92
- "RemoteUser class (Java)" on page 82
- "ServerInfo class (Java)" on page 34
- "UnfencedServer class (Java)" on page 75
- "WrapperInfo class (Java)" on page 48
- "Wrapper class (Java)" on page 56
- "WrapperException class (Java)" on page 190

FencedServer class (Java)

This topic describes the FencedServer class and provides details for the methods.

This class does not contain constructors.

Overview

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

The FencedServer class is one of the server classes for the Java API.

Usage Do not use this class directly. Instantiating or subclassing the FencedServer class directly results in incorrect wrapper behavior. Subclass the FencedGenericServer class.

Package
com.ibm.db2.wrapper

Methods

The following table describes the methods of the FencedServer class. The methods are described in more detail after the table.

Table 85. Methods for the FencedServer class

Methods	Description
createRemoteConnection	Create a new connection of the specified kind for the specified user mapping.
findConnection	Retrieve the current active connection for the data source server.
getRemoteConnectionKind	Return the type of connection that this data source server supports.

createRemoteConnection method

Purpose

Create a new connection of the specified kind for the specified user mapping. The wrapper writer implements this method.

Syntax

```
protected RemoteConnection createRemoteConnection(FencedRemoteUser user,
                                                  int kind,
                                                  long id)
                                           throws java.lang.Exception
```

Parameters

Table 86. Parameters for the createRemoteConnection method

Name	Description
user	User mapping object that is used to authenticate the user.
kind	Connection type.
id	Integer value that represents the RemoteConnection object.

Return value

The newly created connection instance.

Throws

An Exception object if the RemoteConnection instance cannot be created.

findConnection method

Purpose

Retrieve the current active connection for the data source server.

Syntax

```
public final RemoteConnection findConnection()
```

Return value

The active connection for this data source server, or null if no active connection is found.

Throws

None.

getRemoteConnectionKind method

Purpose

Retrieve the type of connection that this data source server supports, as indicated by the constants RemoteConnection.NO_PHASE_KIND or

RemoteConnection.ONE_PHASE_KIND. A default implementation returns the RemoteConnection.NO_PHASE_KIND constant. The subclasses can overwrite this method if necessary.

Syntax

```
protected int getRemoteConnectionKind()
```

Parameters

None.

Return value

The connection type.

Throws

None.

Inherited methods

The following methods are inherited from the Server class and can be used with the FencedServer class:

- createNickname
- createRemoteUser
- findRemoteUser
- getInfo
- getName
- getType
- getVersion
- getWrapper
- initializeMyServer

Related reference:

- "FencedGenericServer class (Java)" on page 73
- "RemoteConnection class (Java)" on page 101
- "Server class (Java)" on page 67
- "WrapperException class (Java)" on page 190

FencedGenericServer class (Java)

This topic describes the FencedGenericServer class and provides details for the constructor and the methods.

Overview

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

The FencedGenericServer class is one of the server classes for the Java API.

Usage The wrapper must implement a subclass of the FencedGenericServer class. This class is instantiated by the wrapper in the createServer method of the wrapper-specific subclass of FencedGenericWrapper.

Package

com.ibm.db2.wrapper

Constructor and methods

The following tables describe the constructor and the method of the FencedGenericServer class. The constructor and method are described in more detail after the tables.

Table 87. Constructor for the FencedGenericServer class

Constructor	Description
FencedGenericServer	Construct a FencedGenericServer object for the specified wrapper with the specified name.

Table 88. Method for the FencedGenericServer class

Method	Description
createRemoteUser	Instantiate an appropriate RemoteUser subclass for this data source server.

FencedGenericServer constructor

Purpose

Construct a FencedGenericServer object for the specified wrapper with the specified name.

Syntax

```
protected FencedGenericServer(java.lang.String name,
                              FencedGenericWrapper wrapper)
```

Parameters

Table 89. Parameters for the FencedGenericServer constructor

Name	Description
name	Name of the data source server.
wrapper	Wrapper object that contains the data source server.

createRemoteUser method

Purpose

Instantiate an appropriate RemoteUser subclass for the specified data source server. By default, the createRemoteUser method creates an instance of the FencedGenericRemoteUser class.

Usage The wrapper can implement this method in the wrapper-specific data source server subclass.

The wrapper writer must implement this method if a wrapper-specific subclass of the FencedGenericRemoteUser class is implemented.

Syntax

```
protected RemoteUser createRemoteUser(java.lang.String userName)
                                throws java.lang.Exception
```

Parameters

Table 90. Parameters for the createRemoteUser method

Name	Description
userName	The name of the remote user mapping to be created, as specified in the CREATE USER MAPPING statement.

Returns

The new FencedGenericRemoteUser instance.

Throws

An Exception object if a new RemoteUser instance cannot be created.

Inherited methods

The following methods are inherited from the FencedServer class and can be used with the FencedGenericServer class:

- createRemoteConnection
- findConnection
- getRemoteConnectionKind

The following methods are inherited from the Server class and can be used with the FencedGenericServer class:

- createNickname
- findRemoteUser
- getInfo
- getName
- getType
- getVersion
- getWrapper
- initializeMyServer

Related tasks:

- "Server classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "FencedGenericWrapper class (Java)" on page 60
- "FencedServer class (Java)" on page 71
- "Server class (Java)" on page 67
- "WrapperException class (Java)" on page 190

UnfencedServer class (Java)

This topic describes the UnfencedServer class and provides details for the methods.

This class does not contain constructors.

Overview

The UnfencedServer class is a subclass of the Server class and is the abstract base class for all data source server functionality that operates in the unfenced (trusted)

process space. This class performs query planning activities. The UnfencedServer class is also responsible for validating the information that is specified in the CREATE NICKNAME and ALTER NICKNAME statements.

The UnfencedServer class is one of the server classes for the Java API.

Usage Do not use this class directly. Instantiating or subclassing the UnfencedServer class directly results in incorrect wrapper behavior. Subclass the UnfencedGenericServer class.

Package
com.ibm.db2.wrapper

Methods

The following table describes the methods of the UnfencedServer class. The methods are described in more detail after the table.

Table 91. Methods for the UnfencedServer class

Method	Description
findNickname	Search for a nickname with the specified schema name and nickname.
verifyMyAlterServerInfo	Validate the data source server information that is specified in ALTER SERVER statements.
verifyMyRegisterServerInfo	Validate the data source server information that is specified in CREATE SERVER statements.

findNickname method

Purpose
Search for a nickname with the specified schema name and nickname.

Syntax

```
public final Nickname findNickname(java.lang.String schema,
                                   java.lang.String name)
```

Parameters

Table 92. Parameters for the findNickname method

Name	Description
schema	Schema name of the nickname to locate.
name	Name of the nickname to locate.

Return value
The nickname with the specified name, or null if the nickname is not found.

Throws
None.

verifyMyAlterServerInfo method

Purpose
Validate the data source server information that is specified in ALTER SERVER statements. By default, this method allows only reserved options

and does not return additional information. The wrapper can implement the verifyMyAlterServerInfo method in the wrapper-specific unfenced server subclass. The wrapper must implement this method if wrapper-specific data source server options are supported.

Syntax

```
protected ServerInfo verifyMyAlterServerInfo(ServerInfo serverInfo)
                                        throws java.lang.Exception
```

Parameters

Table 93. Parameters for the verifyMyAlterServerInfo method

Name	Description
serverInfo	Object that contains the information that is provided in ALTER SERVER statements.

Return value

An object with the information that is added by the data source server.

Throws

An Exception object if the verification process fails.

verifyMyRegisterServerInfo method

Purpose

Validate the server information that is specified in CREATE SERVER statements. By default, this method allows only reserved options and does not return additional information. The wrapper can implement the verifyMyRegisterServerInfo method in the wrapper-specific unfenced server subclass. The wrapper must implement this method if wrapper-specific data source server options are supported.

Syntax

```
protected ServerInfo verifyMyRegisterServerInfo(ServerInfo serverInfo)
                                           throws java.lang.Exception
```

Parameters

Table 94. Parameters for the verifyMyRegisterServerInfo method

Name	Description
serverInfo	Object that contains the information that is provided in CREATE SERVER statements.

Return value

An object with the information that is added by the data source server.

Throws

An Exception object if the verification process fails.

Inherited methods

The following methods are inherited from the Server class and can be used with the UnfencedServer class:

- createNickname
- createRemoteUser

- findRemoteUser
- getInfo
- getName
- getType
- getVersion
- getWrapper
- initializeMyServer

Related reference:

- "ServerInfo class (Java)" on page 34
- "Server class (Java)" on page 67
- "WrapperException class (Java)" on page 190

UnfencedGenericServer class (Java)

This topic describes the UnfencedGenericServer class and provides details for the constructor and the methods.

Overview

The UnfencedGenericServer class is a subclass of the Server class and is the abstract base class for all data source server functionality that operates in the unfenced (trusted) process space. This class performs query planning activities. The UnfencedGenericServer class is also responsible for validating the information that is specified in the CREATE NICKNAME and ALTER NICKNAME statements.

The UnfencedGenericServer class is one of the server classes for the Java API.

Usage The wrapper must implement a subclass of the UnfencedGenericServer class. This class is instantiated by the wrapper in the createServer method of the wrapper-specific subclass of the UnfencedGenericWrapper class.

Package
com.ibm.db2.wrapper

Constructors and methods

The following tables describe the constructor and the methods of the UnfencedGenericServer class. The constructor and methods are described in more detail after the tables.

Table 95. Constructors for the UnfencedGenericServer class

Constructor	Description
UnfencedGenericServer	Construct a UnfencedGenericServer object for the specified wrapper with the specified name.

Table 96. Methods for the UnfencedGenericServer class

Method	Description
createRemoteUser	Instantiate an appropriate RemoteUser subclass for this data source server.
createReply	Create a new empty Reply object for the given Request.

Table 96. Methods for the UnfencedGenericServer class (continued)

Method	Description
getSelectivity	Calculate the selectivity of a list of predicates.
planRequest	Analyze a proposed plan and determine what part of the plan (if any) can be pushed down to the remote data source.

UnfencedGenericServer constructor

Purpose

Constructs a UnfencedGenericServer object for the specified wrapper with the specified name.

Syntax

```
protected UnfencedGenericServer(java.lang.String name,
                                UnfencedGenericWrapper wrapper)
```

Parameters

Table 97. Parameters for the UnfencedGenericServer constructor

Name	Description
name	Name of the data source server.
wrapper	Wrapper object that contains the data source server.

Throws

None.

createRemoteUser method

Purpose

Instantiate an appropriate RemoteUser subclass for this data source server. By default, this method creates an instance of the UnfencedGenericRemoteUser class.

Usage The wrapper can implement this method in the wrapper-specific server subclass. If a wrapper-specific subclass of the UnfencedGenericRemoteUser class is implemented, this method must also be implemented.

Syntax

```
protected RemoteUser createRemoteUser(java.lang.String userName)
                                throws java.lang.Exception
```

Parameters

Table 98. Parameters for the createRemoteUser method

Name	Description
userName	Name of the remote user mapping to be created and that is specified in the CREATE USER MAPPING statement.

Return value

The newly created RemoteUser instance.

Throws

An Exception object if the RemoteUser instance cannot be created.

createReply method

Purpose

Create a new empty Reply object for the given Request. The planRequest method invokes this method to create the Reply objects.

Syntax

```
public final Reply createReply(Request request)
                        throws java.lang.Exception
```

Parameters

Table 99. Parameters for the createReply method

Name	Description
request	Request object that the Reply object is created for.

Return value

The newly created Reply instance.

Throws

An Exception object if the method fails.

getSelectivity method

Purpose

Calculate the selectivity of a list of predicates.

Syntax

```
public float getSelectivity(PredicateList predicateList)
                     throws java.lang.Exception
```

Parameters

Table 100. Parameters for the getSelectivity method

Name	Description
PredicateList	PredicateList instance.

Return value

The selectivity value.

Throws

An Exception object if the method fails.

planRequest method

Purpose

Analyze a proposed plan and determine what part of the plan (if any) can be pushed down to the remote data source. The wrapper writer implements this method.

Syntax

```
public Reply planRequest(Request request)
                  throws java.lang.Excepttion
```

Parameters

Table 101. Parameters for the planRequest method

Name	Description
Request	The Request object that contains the planned query.

Return value

The Reply object or the first Reply in a list of Replies. Each Reply describes the query fragment that the wrapper is able to push down to the remote data source.

Throws

An Exception object if the method fails.

Inherited methods

The following methods are inherited from the UnfencedServer class and can be used with the UnfencedGenericServer class:

- findNickname
- verifyMyAlterServerInfo,
- verifyMyRegisterServerInfo

The following methods are inherited from the Server class and can be used with the UnfencedGenericServer class:

- createNickname
- dropAllNicknames
- findRemoteUser
- getInfo
- getName
- getType
- getVersion
- getWrapper
- initializeMyServer

Related tasks:

- "Server classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "PredicateList class (Java)" on page 154
- "Reply class (Java)" on page 131
- "Request class (Java)" on page 130
- "RemoteUser class (Java)" on page 82
- "Server class (Java)" on page 67
- "UnfencedServer class (Java)" on page 75
- "WrapperException class (Java)" on page 190

User classes for the Java API

The following table describes each user class for the Java API.

Table 102. User classes

Class name	Description
RemoteUser	The class that represents the authorizations to use on a data source server that the wrapper communicates with.
FencedRemoteUser	The abstract base class that represents a user mapping in the fenced (untrusted) process space. Do not use this class directly. Instantiating or subclassing the FencedRemoteUser class directly results in incorrect wrapper behavior. Use the FencedGenericRemoteUser class.
FencedGenericRemoteUser	The class that represents a user mapping for a data source in the fenced (untrusted) process space.
UnfencedRemoteUser	The abstract base class that represents a user mapping in the unfenced (trusted) process space. Do not use this class directly. Instantiating or subclassing the UnfencedRemoteUser class directly results in incorrect wrapper behavior. Use the UnfencedGenericRemoteUser class.
UnfencedGenericRemoteUser	The class that represents a user mapping for a data source in the unfenced (trusted) process space and is instantiated to validate the information that is specified in the CREATE USER MAPPING and ALTER USER MAPPING statements.

Related reference:

- "FencedGenericRemoteUser class (Java)" on page 86
- "FencedRemoteUser class (Java)" on page 85
- "RemoteUser class (Java)" on page 82
- "UnfencedGenericRemoteUser class (Java)" on page 89
- "UnfencedRemoteUser class (Java)" on page 87

RemoteUser class (Java)

This topic describes the RemoteUser class and provides details for the methds.

This class does not contain constructors.

Overview

The RemoteUser class represents the authorizations to use on a data source server. Each instance of this class or its subclasses represents an authorization to use on the data source server that the wrapper works with. The RemoteUser base class implementation maintains the following information:

- The local authorization ID of the user
- A UserInfo object that contains information about the data source server and user pair that was stored in the federated server's system catalog as a result of running DDL statements
- A reference to the data source server object that contains the user information

The FencedRemoteUser class and the UnfencedRemoteUser class are subclasses of the RemoteUser class.

The RemoteUser class is one of the user classes for the Java API.

Usage Do not use this class directly. Instantiate or subclass the FencedGenericRemoteUser class and the UnfencedGenericRemoteUser class.

Package
com.ibm.db2.wrapper

Constants and class fields

The following table describes the valid constants and class fields for the options that you can use with the RemoteUser class.

Table 103. Constants and class fields for the RemoteUser class

Constant or class field	Definition	Description
REMOTE_AUTHID_ OPTION	public static final java.lang.String REMOTE_AUTHID_ OPTION	Constant that represents the name of the remote authorization ID option.
REMOTE_ PASSWORD_OPTION	public static final java.lang.String REMOTE_PASSWORD _OPTION	Constant that represents the name of the remote authorization password option.

Methods

The following table describes the methods of the RemoteUser class. The methods are described in more detail after the table.

Table 104. Methods for the RemoteUser class

Method	Description
getInfo	Retrieve the user mapping information that is stored in the federated server's system catalog as a result of running DDL statements.
getLocalName	Retrieve the name of the user on the local database.
getWrapper	Retrieve the wrapper instance that this user mapping belongs to.
initializeMyUser	Perform the necessary user mapping initialization.

getInfo method

Purpose
Retrieve the user mapping information that is stored in the federated server's system catalog as a result of running DDL statements.

Syntax

```
public final UserInfo getInfo()
```

Parameters
None.

Return value
An instance of UserInfo that contains the user mapping information.

Throws
None.

getLocalName method

Purpose
Retrieve the name of the user on the local database.

Syntax

```
public final java.lang.String getLocalName()
```

Parameters
None.

Return value
The local user name.

Throws
None.

getWrapper method

Purpose
Retrieve the wrapper instance that this user belongs to.

Syntax

```
public final Wrapper getWrapper()
```

Parameters
None.

Return value
The wrapper object.

Throws
None.

initializeMyUser method

Purpose
Perform the necessary user mapping initialization. This method is called when the user mapping is created or when its options are changed.

This method can be implemented by the wrapper writer in the wrapper-specific RemoteUser subclass to perform the user mapping initialization.

Syntax

```
protected void initializeMyUser(UserInfo userInfo)
                          throws java.lang.Exception
```

Parameters

Table 105. Parameters for the initializeMyUser method

Name	Description
userInfo	Instance of UserInfo that represents the user mapping information.

Return value
None.

Throws

An Exception object if the initialization process fails.

Related reference:

- "Server class (Java)" on page 67
- "UserInfo class (Java)" on page 44
- "WrapperException class (Java)" on page 190

FencedRemoteUser class (Java)

This topic describes the FencedRemoteUser class.

Overview

The FencedRemoteUser class is the abstract base class that represents a user mapping in the fenced (untrusted) process space.

The public FencedRemoteUser class extends the RemoteUser class. The FencedGenericRemoteUser class is a subclass of the FencedRemoteUser class.

The FencedRemoteUser class is one of the user classes for the Java API.

Usage Do not use the FencedRemoteUser class directly. Instantiating or subclassing the FencedRemoteUser class directly results in incorrect wrapper behavior. Instantiate or subclass the FencedGenericRemoteUser class.

Package

com.ibm.db2.wrapper

Constants and class fields

The following table describes the valid constants and class fields for the options that you can use with the FencedRemoteUser class. These constants and class fields are inherited from the RemoteUser class.

Table 106. Constants and class fields for the FencedRemoteUser class

Constant or class field	Definition	Description
REMOTE_AUTHID_ OPTION	public static final java.lang.String REMOTE_AUTHID_ OPTION	Constant that represents the name of the remote authorization ID option.
REMOTE_ PASSWORD_OPTION	public static final java.lang.String REMOTE_PASSWORD _OPTION	Constant that represents the name of the remote authorization password option.

Inherited methods

The following methods are inherited from the RemoteUser class and can be used with the FencedRemoteUser class:

- getInfo
- getLocalName
- getWrapper
- initializeMyUser

Related reference:

- "FencedServer class (Java)" on page 71
- "RemoteUser class (Java)" on page 82
- "WrapperException class (Java)" on page 190

FencedGenericRemoteUser class (Java)

This topic describes the FencedGenericRemoteUser class.

Overview

The FencedGenericRemoteUser class represents a user mapping for a data source in the fenced (untrusted) process space.

The FencedGenericRemoteUser class is one of the user classes for the Java API.

Usage This class is instantiated by the wrapper in the createRemoteUser method of the wrapper-specific subclass of FencedGenericServer. The wrapper implements a subclass of FencedGenericRemoteUser if wrapper-specific user mapping options are supported.

Package

com.ibm.db2.wrapper

Constants and class fields

The following table describes the valid constants and class fields for the options that you can use with the FencedGenericRemoteUser class. These constants and class fields are inherited from the RemoteUser class.

Table 107. Constants and class fields for the FencedGenericRemoteUser class

Constant or class field	Definition	Description
REMOTE_AUTHID_ OPTION	public static final java.lang.String REMOTE_AUTHID_ OPTION	Constant that represents the name of the remote authorization ID option.
REMOTE_ PASSWORD_OPTION	public static final java.lang.String REMOTE_PASSWORD _OPTION	Constant that represents the name of the remote authorization password option.

Constructor

The following table describes the constructor of the FencedGenericRemoteUser class. The constructor is described in more detail after the table.

Table 108. Constructor for the FencedGenericRemoteUser class

Constructor	Description
FencedGenericRemoteUser	Construct a user mapping with a specified name for the specified data source server.

FencedGenericRemoteUser constructor

Purpose

Construct a user mapping with a specified name for the specified data source server.

Syntax

```
public FencedGenericRemoteUser(java.lang.String localName,
                               FencedGenericServer server)
```

Parameters

Table 109. Parameters for the FencedGenericRemoteUser constructor

Name	Description
localName	Local DB2 Information Integrator name of the user mapping.
server	Server object that contains the user mapping.

Throws

None.

Inherited methods

The following methods are inherited from the RemoteUser class and can be used with the FencedGenericRemoteUser class:

- getInfo
- getLocalName
- getWrapper
- initializeMyUser

Related tasks:

- "User classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "FencedGenericServer class (Java)" on page 73
- "FencedRemoteUser class (Java)" on page 85
- "RemoteUser class (Java)" on page 82
- "WrapperException class (Java)" on page 190

UnfencedRemoteUser class (Java)

This topic describes the UnfencedRemoteUser class and provides details for the methods.

This class does not contain constructors.

Overview

The UnfencedRemoteUser class is the abstract base class that represents a user mapping in the unfenced (trusted) process space. This class is instantiated to validate the information that is specified in the CREATE USER MAPPING and ALTER USER MAPPING statements.

The public UnfencedRemoteUser class extends the RemoteUser class. The UnfencedGenericRemoteUser class is a subclass of the UnfencedRemoteUser class.

The UnfencedRemoteUser class is one of the user classes for the Java API.

Usage Do not use the UnfencedRemoteUser class directly. Instantiating or

subclassing the UnfencedRemoteUser class directly results in incorrect wrapper behavior. Instantiate or subclass the UnfencedGenericRemoteUser class.

Package

com.ibm.db2.wrapper

Constants and class fields

The following table describes the valid constants and class fields for the options that you can use with the UnfencedRemoteUser class. These constants and class fields are inherited from the RemoteUser class.

Table 110. Constants and class fields for the UnfencedRemoteUser class

Constant or class field	Definition	Description
REMOTE_AUTHID_ OPTION	public static final java.lang.String REMOTE_AUTHID_ OPTION	Constant that represents the name of the remote authorization ID option.
REMOTE_ PASSWORD_OPTION	public static final java.lang.String REMOTE_PASSWORD _OPTION	Constant that represents the name of the remote authorization password option.

Methods

The following table describes the methods of the UnfencedRemoteUser class. The methods are described in more detail after the table.

Table 111. Methods for the UnfencedRemoteUser class

Method	Description
verifyMyAlterUserInfo	Validate the user mapping information that is specified in ALTER USER MAPPING statements.
verifyMyRegisterUserInfo	Validate the user mapping information that is specified in the CREATE USER MAPPING statements.

verifyMyAlterUserInfo method

Purpose

Validate the user mapping information that is specified in ALTER USER MAPPING statements. By default, this method allows only reserved options and does not return other options. This method must be implemented if wrapper-specific user mapping options are supported.

Syntax

```
protected UserInfo verifyMyAlterUserInfo(UserInfo userInfo)
                                    throws java.lang.Exception
```

Parameters

Table 112. Parameters for the verifyMyAlterUserInfo method

Name	Description
userInfo	UserInfo object that contains the information that is provided in ALTER USER MAPPING statements.

Return value

A UserInfo object with the information that was added by the wrapper.

Throws

An Exception object if the verification process fails.

verifyMyRegisterUserInfo method

Purpose

Validate the user mapping information that is specified in the CREATE USER MAPPING statements. By default, this method allows only reserved options and does not return other options. This method must be implemented if wrapper-specific user mapping options are supported.

Syntax

```
protected UserInfo verifyMyRegisterUserInfo(UserInfo userInfo)
                                        throws java.lang.Exception
```

Parameters

Table 113. Parameters for the verifyMyRegisterUserInfo method

Name	Description
userInfo	UserInfo object that contains the information that is provided in CREATE USER MAPPING statements.

Return value

A UserInfo object with the information that was added by the wrapper.

Throws

An Exception object if the verification process fails.

Inherited methods

The following methods are inherited from the RemoteUser class and can be used with the UnfencedRemoteUser class:

- getInfo
- getLocalName
- getWrapper
- initializeMyUser

Related reference:

- "RemoteUser class (Java)" on page 82
- "UserInfo class (Java)" on page 44
- "WrapperInfo class (Java)" on page 48
- "WrapperException class (Java)" on page 190

UnfencedGenericRemoteUser class (Java)

This topic describes the UnfencedGenericRemoteUser class.

Overview

The UnfencedGenericRemoteUser class represents a user mapping for a data source in the unfenced (trusted) process space.

The UnfencedGenericRemoteUser class is one of the user classes for the Java API.

Usage This class is instantiated by the wrapper in the createRemoteUser method of the wrapper-specific subclass of the UnfencedGenericServer class. The wrapper must implement a subclass of UnfencedGenericRemoteUser if wrapper-specific user mapping options for the CREATE USER MAPPING or the ALTER USER MAPPING statement are used.

Package

com.ibm.db2.wrapper

Constants and class fields

The following table describes the valid constants and class fields for the options that you can use with the UnfencedGenericRemoteUser class. These constants and class fields are inherited from the RemoteUser class.

Table 114. Constants and class fields for the UnfencedGenericRemoteUser class

Constant or class field	Definition	Description
REMOTE_AUTHID_ OPTION	public static final java.lang.String REMOTE_AUTHID_ OPTION	Constant that represents the name of the remote authorization ID option.
REMOTE_ PASSWORD_OPTION	public static final java.lang.String REMOTE_PASSWORD _OPTION	Constant that represents the name of the remote authorization password option.

Constructor

The following table describes the constructor of the UnfencedGenericRemoteUser class. The constructor is described in more detail after the table.

Table 115. Constructor for the UnfencedGenericRemoteUser class

Constructor	Description
UnfencedGenericRemoteUser	Construct a user mapping with a specified name for the specified data source server.

UnfencedGenericRemoteUser constructor

Purpose

Construct a user mapping with a specified name for the specified data source server.

Syntax

```
public UnfencedGenericRemoteUser(java.lang.String localName,
                                 UnfencedGenericServer server)
```

Parameters

Table 116. Parameters for the UnfencedGenericRemoteUser constructor

Name	Description
localName	Local DB2 Information Integrator name for the user mapping.
server	Server object that contains the user mapping.

Throws

None.

Inherited methods

The following methods are inherited from the UnfencedRemoteUser class and can be used with the UnfencedGenericRemoteUser class:

- verifyMyAlterUserInfo
- verifyMyRegisterUserInfo

The following methods are inherited from the RemoteUser class and can be used with the UnfencedGenericRemoteUser class:

- getInfo
- getLocalName
- getWrapper
- initializeMyUser

Related tasks:

- "User classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "RemoteUser class (Java)" on page 82
- "UnfencedGenericServer class (Java)" on page 78
- "UnfencedRemoteUser class (Java)" on page 87
- "WrapperException class (Java)" on page 190

Nickname classes for the Java API

The following table describes each nickname class for the Java API.

Table 117. Server classes

Class name	Description
Nickname	The class that represents a collection of data that is managed by a data source server that the wrapper works with.
FencedNickname	The abstract base class that represents a nickname in the fenced (untrusted) process space. Do not use this class directly. Instantiating or subclassing the FencedNickname class directly results in incorrect wrapper behavior. Subclass the FencedGenericNickname class.
FencedGenericNickname	The class that represents a nickname in the fenced (untrusted) process space and is responsible for validating information from the CREATE NICKNAME statement.
UnfencedNickname	The abstract base class that represents a nickname in the unfenced (trusted) process space. Do not use this class directly. Instantiating or subclassing the UnfencedNickname class directly results in incorrect wrapper behavior. Subclass the UnfencedGenericNickname class.
UnfencedGenericNickname	The class that represents a nickname in the unfenced (trusted) process space. This class is responsible for validating information from the CREATE NICKNAME and ALTER NICKNAME statements.

Related reference:

- "FencedGenericNickname class (Java)" on page 96
- "FencedNickname class (Java)" on page 95
- "Nickname class (Java)" on page 92
- "UnfencedGenericNickname class (Java)" on page 99
- "UnfencedNickname class (Java)" on page 98

Nickname class (Java)

This topic describes the Nickname class and provides details for the methods.

This class does not contain constructors.

Overview

The Nickname class models collections of data that a data source server manages. Each instance of this class or its subclasses represent a collection of data that is managed by the data source server that the wrapper works with. The Nickname base class implementation maintains the following information:

- The local name for the remote data set that is defined for the nickname.
- The local schema for the remote data set that is defined for the nickname.
- A NicknameInfo object that contains information about a nickname that was stored in the federated server's system catalog after running the CREATE NICKNAME or ALTER NICKNAME statement. The NicknameInfo object references ColumnInfo objects that contain the local name of each column and additional information that was stored in the federated server's system catalog as a result of executing DDL statements.
- A reference to the data source server object that contains this nickname.

The FencedNickname class and UnfencedNickname class are subclasses of the Nickname class.

The Nickname class is one of the user classes for the Java API.

Usage Do not use the Nickname class directly, but subclass the FencedGenericNickname class and the UnfencedGenericNickname class.

Package
com.ibm.db2.wrapper

Methods

The following table describe the methods of the Nickname class. The methods are described in more detail after the table.

Table 118. Methods for the Nickname class

Method	Description
getInfo	Retrieve the nickname information that is stored in the federated server's system catalog as a result of running DDL statements.
getLocalName	Retrieve the local name for this nickname.
getLocalSchema	Retrieve the local schema for this nickname.
getServer	Retrieve the data source server that contains this nickname.

Table 118. Methods for the Nickname class (continued)

Method	Description
getWrapper	Retrieve the wrapper instance that this nickname belongs to.
initializeMyNickname	Perform the necessary nickname initialization.
verifyMyRegisterNicknameInfo	Validate the nickname information that is specified in CREATE NICKNAME statements.

getInfo method

Purpose

Retrieve the nickname information that is stored in the federated server's system catalog as a result of running DDL statements.

Syntax

```
public final NicknameInfo getInfo()
```

Parameters

None.

Return value

The NicknameInfo instance that contains the nickname information.

Throws

None.

getLocalName method

Purpose

Retrieve the local name of this nickname.

Syntax

```
public final java.lang.String getLocalName()
```

Parameters

None.

Return value

The local name of the nickname.

Throws

None.

getLocalSchema method

Purpose

Retrieve the local schema of this nickname.

Syntax

```
public final java.lang.String getLocalSchema()
```

Parameters

None.

Return value

The local schema of the nickname.

Throws

None.

getServer method

Purpose

Retrieve the data source server that contains this nickname.

Syntax

```
public final Server getServer()
```

Parameters

None.

Return value

The data source server.

Throws

None.

getWrapper method

Purpose

Retrieve the wrapper instance that this nickname belongs to.

Syntax

```
public final Wrapper getWrapper()
```

Parameters

None.

Return value

The wrapper object.

Throws

None.

initializeMyNickname method

Purpose

Perform necessary nickname initialization. This method is called when the nickname is created or when the options are changed. The wrapper writer can implement this method in the wrapper-specific nickname to invoke the nickname-specific initialization process.

Syntax

```
protected void initializeMyNickname(NicknameInfo nicknameInfo)
                                throws java.lang.Exception
```

Parameters

Table 119. Parameters for the initializeMyNickname method

Name	Description
nicknameInfo	Nickname catalog information.

Return value

None.

Throws

An Exception object if the initialization process fails.

verifyMyRegisterNicknameInfo method

Purpose

Validate the nickname information that is specified in CREATE

NICKNAME statements. By default, this method allows only reserved options and does not return other options.

Usage The wrapper can implement the verifyMyRegisterNicknameInfo method in the wrapper-specific nickname subclass. Either the unfenced class or the fenced class must implement this method if the wrapper-specific nickname or column options are supported. Because the verifyMyRegisterNicknameInfo method of the UnfencedGenericNickname class is part of the trusted process space, the method cannot interact with the remote data source. If interaction with the remote data source is necessary to verify the nickname information, the verifyMyRegisterNicknameInfo method of the FencedGenericNickname class must be implemented.

Syntax

```
protected NicknameInfo verifyMyRegisterNicknameInfo(NicknameInfo nicknameInfo)
                                                    throws java.lang.Exception
```

Parameters

Table 120. Parameters for the verifyMyRegisterNicknameInfo method

Name	Description
nicknameInfo	The information from the CREATE NICKNAME statement.

Return value
The information that is added by the nickname object.

Throws
An Exception object if the verification process fails.

Related reference:

- "NicknameInfo class (Java)" on page 24
- "Server class (Java)" on page 67
- "WrapperException class (Java)" on page 190

FencedNickname class (Java)

This topic describes the FencedNickname class.

Overview

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

The public FencedGenericServer class extends the FencedServer class. The FencedGenericNickname class is a subclass of the FencedNickname class.

The FencedNickname class is one of the server classes for the Java API.

Usage Do not use the FencedNickname class directly. Instantiating or subclassing the FencedNickname class directly results in incorrect wrapper behavior. Subclass the FencedGenericNickname class.

Package
com.ibm.db2.wrapper

Inherited methods

The following methods are inherited from the Nickname class and can be used with the FencedNickname class:

- getInfo
- getLocalName
- getLocalSchema
- getServer
- getWrapper
- initializeMyNickname
- verifyMyRegisterNicknameInfo

Related reference:

- "NicknameInfo class (Java)" on page 24
- "Nickname class (Java)" on page 92
- "WrapperException class (Java)" on page 190

FencedGenericNickname class (Java)

This topic describes the FencedGenericNickname class.

Overview

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

The FencedGenericNickname class extends the FencedNickname class.

The FencedGenericNickname class is one of the user classes for the Java API.

Usage The wrapper must subclass the FencedGenericNickname class. The FencedGenericNickname class is instantiated by the wrapper in the createNickname method of the wrapper-specific subclass of FencedGenericServer.

Package
com.ibm.db2.wrapper

Constructor

The following table describes the constructor of the FencedGenericNickname class. The constructor is described in more detail after the table.

Table 121. Constructors for the FencedGenericNickname class

Constructor	Description
FencedGenericNickname	Construct a nickname with the specified schema and name for the specified data source server.

FencedGenericNickname constructor

Purpose

Construct a nickname with the specified schema and name for the specified data source server.

Syntax

```
protected FencedGenericNickname(java.lang.String schema,
                                java.lang.String name,
                                FencedGenericServer server)
```

Parameters

Table 122. Parameters for the FencedGenericNickname constructor

Name	Description
schema	Local DB2 Information Integrator schema for the remote data set that is defined for this nickname.
name	Local DB2 Information Integrator name for the remote data set that is defined for this nickname.
server	Server object that contains the nickname.

Throws

None.

Inherited methods

The following methods are inherited from the Nickname class and can be used with the FencedGenericNickname class:

- getInfo
- getLocalName
- getLocalSchema
- getServer
- getWrapper
- initializeMyNickname
- verifyMyRegisterNicknameInfo

Related tasks:

- "Nickname classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "FencedGenericServer class (Java)" on page 73
- "FencedNickname class (Java)" on page 95
- "Nickname class (Java)" on page 92
- "WrapperException class (Java)" on page 190

UnfencedNickname class (Java)

This topic describes the UnfencedNickname class and provides details for the constructor and the methods.

This class does not contain any constructors.

Overview

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

The UnfencedNickname class extends the Nickname class.

The UnfencedNickname class is one of the user classes for the Java API.

Usage Do not use the UnfencedNickname class directly. Instantiating or subclassing the UnfencedNickname class directly results in incorrect wrapper behavior. Subclass the UnfencedGenericNickname class.

Package

com.ibm.db2.wrapper

Method

The following table describes the method of the UnfencedNickname class. The method is described in more detail after the table.

Table 123. Method for the UnfencedNickname class

Method	Description
verifyMyAlterNicknameInfo	Validate the nickname information that is specified in ALTER NICKNAME statements.

verifyMyAlterNicknameInfo method

Purpose

Validate the nickname information that is specified in ALTER NICKNAME statements. By default, this method allows only reserved options and does not return other options.

Usage The wrapper can implement the verifyMyAlterNicknameInfo method in the wrapper-specific nickname subclass. The verifyMyAlterNicknameInfo method must be implemented if the wrapper-specific nickname or column options are supported. Because the verifyMyAlterNicknameInfo method is part of the trusted process space, the method cannot interact with the remote data source.

Syntax

```
protected NicknameInfo verifyMyAlterNicknameInfo(NicknameInfo nicknameInfo)
                                          throws java.lang.Exception
```

Parameters

Table 124. Parameters for the verifyMyAlterNicknameInfo method

Name	Description
nicknameInfo	Object that contains the information that is provided in an ALTER NICKNAME statement.

Return value
An object with the information that is added by the nickname.

Throws
An Exception object if the verification process fails.

Inherited methods

The following methods are inherited from the Nickname class and can be used with the UnfencedNickname class:

- getInfo
- getLocalName
- getLocalSchema
- getServer
- getWrapper
- initializeMyNickname
- verifyMyRegisterNicknameInfo

Related reference:

- "NicknameInfo class (Java)" on page 24
- "Nickname class (Java)" on page 92
- "WrapperException class (Java)" on page 190

UnfencedGenericNickname class (Java)

This topic describes the UnfencedGenericNickname class.

Overview

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

The UnfencedGenericNickname class extends the UnfencedNickname class.

The UnfencedGenericNickname class is one of the user classes for the Java API.

Usage The UnfencedGenericNickname class must be subclassed by the wrapper and is instantiated by the wrapper in the createNickname method of the wrapper-specific subclass of UnfencedGenericServer.

Package
com.ibm.db2.wrapper

Constructor

The following table describes the constructor of the UnfencedGenericNickname class. The constructor is described in more detail after the table.

Table 125. Constructors for the UnfencedGenericNickname class

Constructor	Description
UnfencedGenericNickname	Construct a nickname with the specified schema and name for the specified data source server.

UnfencedGenericNickname constructor

Purpose

Construct a nickname with the specified schema and name for the specified data source server.

Syntax

```
protected UnfencedGenericNickname(java.lang.String schema,
                                  java.lang.String name,
                                  UnfencedGenericServer server)
```

Parameters

Table 126. Parameters for the UnfencedGenericNickname constructor

Name	Description
schema	Local DB2 Information Integrator schema for the remote data set that is defined for this nickname.
name	Local DB2 Information Integrator name for the remote data set that is defined for this nickname.
server	Server object that contains the nickname.

Throws

None.

Inherited methods

The verifyMyAlterNicknameInfo method is inherited from the UnfencedNickname class and also can be used with the UnfencedGenericNickname class.

The following methods are inherited from the Nickname class and can be used with the UnfencedGenericNickname class:

- getInfo
- getLocalName
- getLocalSchema
- getServer
- getWrapper
- initializeMyNickname
- verifyMyRegisterNicknameInfo

Related tasks:

- "Nickname classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "Nickname class (Java)" on page 92
- "UnfencedNickname class (Java)" on page 98
- "WrapperException class (Java)" on page 190

RemoteConnection class (Java)

This topic describes the RemoteConnection class and provides details for the constructor and the methods.

Overview

The RemoteConnection class represents a connection (session) with a data source server. The operations on the data source server (database) are processed through a connection. Connections are created only for untrusted and fenced data source servers. The RemoteConnection base class implementation maintains the following information:

- The status (connected or disconnected) of your connection to your data source.
- A reference to the appropriate FencedServer object.
- A reference to the appropriate FencedRemoteUser object.
- The code page to use for this connection.
- The connection type. See Table 127for more information about connection types.

The RemoteConnection class is a connection class for the Java API.

Usage You can create an instance of your RemoteConnection subclass by invoking the FencedServer.createRemoteConnection method on an instance of the appropriate fenced server subclass. The federated server calls this method before processing the first remote operation at the relevant data source server. The federated server destroys the RemoteConnection instance after a predefined number of transactions are processed by the application without using the data source server.

Package

com.ibm.db2.wrapper

Constants and class fields

The following table describes the valid constants and class fields for the options that you can use with the RemoteConnection class.

Table 127. Constants and class fields for the RemoteConnection class

Constant or class field	Definition	Description
NO_PHASE_KIND	public static final int NO_PHASE_KIND	Indicates that no transactions are supported.
ONE_PHASE_KIND	public static final int ONE_PHASE_KIND	Indicates that one-phase commit transactions are supported.

Constructor and methods

The following tables describe the constructor and the methods of the RemoteConnection class. The constructor and methods are described in more detail after the tables.

Table 128. Constructors for the RemoteConnection class

Constructor	Description
RemoteConnection	Construct a connection for the specified data source server with the user authorization and transaction type as specified.

Table 129. Methods for the RemoteConnection class

Method	Description
commit	Indicate the successful completion of a transaction, and that the remote data source then commits the transaction.
connect	Invoke the code to connect to the data source server.
createRemotePassthru	Create a RemotePassthru object to run pass-through statements.
createRemoteQuery	Create a RemoteQuery object to run SQL statements.
disconnect	Invoke the code to disconnect from the data source server.
getCodepage	Retrieve the code page for the connection.
getKind	Retrieve the connection type.
getServer	Retrive the data source server object that contains this connection.
getUser	Retrieve the user mapping to authenticate this connection.
getWrapper	Retrieve the wrapper object.
isConnected	Verify whether the connection with the data source server exists.
markDisconnected	Set the flag to indicate that the connection with the data source server finished.
rollback	Indicate the unsuccessful completion of a transaction, and that the remote data source then rolls back the transaction.

RemoteConnection constructor

Purpose

Construct a connection for the specified data source server with the specified user authorization and type.

Syntax

```
protected RemoteConnection(FencedServer remoteServer,
                           FencedRemoteUser remoteUser,
                           int connectionKind,
                           long id)
```

Parameters

Table 130. Parameters for the RemoteConnection constructor

Name	Description
remoteServer	Data source server that contains the connection.

Table 130. Parameters for the RemoteConnection constructor (continued)

Name	Description
remoteUser	User mapping that is used for authentication.
connectionKind	Type of connection. Specifies the supported transaction types. A connection that can support one-phase commit transactions is indicated by the ONE_PHASE_KIND constant. A connection that does not have transaction support is indicated by the NO_PHASE_KIND constant.
id	An integer value that represents the RemoteConnection object.

Throws

None.

commit method

Purpose

Indicate the successful completion of a transaction, and that the remote data source then commits the transaction. A default implementation does nothing. The wrapper can override this method to implement necessary commit logic for the remote data source.

Syntax

```
protected void commit()
            throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the method fails.

connect method

Purpose

Invoke the code to connect to the data source server. This method does nothing by default. The connect method can be implemented if the wrapper-specific connection class needs to perform any operation to establish a connection with the remote data source.

Syntax

```
protected void connect()
             java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the method fails.

createRemotePassthru method

Purpose

Create a RemotePassthru object to run pass-through statements. This method returns null by default. However, if the wrapper supports the ability to run pass-through statements, it must implement the method in the wrapper-specific connection class. When the createRemotePassthru method is used in this situation, it must return a value that is not null.

Syntax

```
protected RemotePassthru createRemotePassthru(long id)
                                        throws java.lang.Exception
```

Parameters

Table 131. Parameters for the createRemotePassthru method

Name	Description
id	An integer value that represents the the RemotePassthru object.

Return value

A RemotePassthru instance or null if the wrapper does not support the operation.

Throws

An Exception object if the object creation fails.

createRemoteQuery method

Purpose

Create a RemoteQuery object to run SQL statements. This method returns null by default. However, if the wrapper supports the ability to run SQL statements, it must implement the method in the wrapper-specific connection class. When the createRemoteQuery method is used in this situation, it must return a value that is not null.

Syntax

```
protected RemoteQuery createRemoteQuery(long id)
                                  throws java.lang.Exception
```

Parameters

Table 132. Parameters for the createRemoteQuery method

Name	Description
id	An integer value that represents the RemoteQuery object

Return value

A RemoteQuery instance or null if the wrapper does not support the operation.

Throws

An Exception object if the object creation fails.

disconnect method

Purpose

Invoke the code to disconnect from the data source server. The disconnect

method can be implemented if the wrapper-specific connection class needs to perform any operation to terminate a connection with the remote data source.

Syntax

```
protected void disconnect()
                throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the method fails.

getCodepage method

Purpose

Retrive the code page for the connection.

Syntax

```
public final short getCodepage()
```

Parameters

None.

Return value

The code page.

Throws

None.

getKind method

Purpose

Retrieve the connection type. A connection can either support one-phase commit transactions (indicated by the ONE_PHASE_KIND constant), or cannot support transactions (indicated by the NO_PHASE_KIND constant).

Syntax

```
public final int getKind()
```

Parameters

None.

Return value

The connection type.

Throws

None.

getServer method

Purpose

Retrieve the data source server object that contains this connection.

Syntax

```
public final FencedServer getServer()
```

Parameters

None.

Return value
The data source server for this connection.

Throws
None.

getUser method

Purpose
Retrieve the user mapping to authenticates this connection.

Syntax

```
public final FencedRemoteUser getUser()
```

Parameters
None.

Return value
The user mapping for this connection.

Throws
None.

getWrapper method

Purpose
Retrieve the wrapper object.

Syntax

```
public final Wrapper getWrapper()
```

Parameters
None.

Return value
The wrapper object.

Throws
None.

isConnected method

Purpose
Verify if a connection with the data source server exists.

Syntax

```
public final boolean isConnected()
```

Parameters
None.

Return value
A value that indicates whether or not a connection with the data source server exists.

Throws
None.

markDisconnected method

Purpose
Set the flag to indicate that the connection with the data source server finished.

Syntax

```
public final void markDisconnected()
```

Parameters
None.

Return value
None.

Throws
None.

rollback method

Purpose
Indicate the unsuccessful completion of a transaction, and that the remote data source then rolls back the transaction. A default implementation does nothing. The wrapper can override this method to implement necessary rollback logic for the remote data source.

Syntax

```
protected void rollback()
             throws java.lang.Exception
```

Parameters
None.

Return value
None.

Throws
An Exception object if the method fails.

Related tasks:

- "Remote connection class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "FencedRemoteUser class (Java)" on page 85
- "FencedServer class (Java)" on page 71
- "RemotePassthru class (Java)" on page 111
- "RemoteQuery class (Java)" on page 115
- "Wrapper class (Java)" on page 56
- "WrapperException class (Java)" on page 190

Operation classes for the Java API

The following table describes each operation class for the Java API.

Table 133. Operation classes

Class name	Description
RemoteOperation	The class that represents the base class for classes that represent various remote operations. These remote operations include queries and pass-through sessions.
RemotePassthru	The class that represents a pass-through session on a remote data source.
RemoteQuery	The class that represents SELECT statement operations on a remote data source.

Related reference:
- "RemoteOperation class (Java)" on page 108
- "RemotePassthru class (Java)" on page 111
- "RemoteQuery class (Java)" on page 115

RemoteOperation class (Java)

This topic describes the RemoteOperation class and provides details for methods.

This class does not contain constructors.

Overview

The RemoteOperation class is the base class for classes that represent various remote operations. These remote operations include queries and pass-through sessions. The RemoteOperation class is not directly instantiated and cannot be customized directly. The RemoteOperation base class implementation maintains the following information:

- The type of remote operation that is represented by the object.
- A reference to the RemoteConnection object.
- A reference to a RuntimeDataList that describes the type and location of values to be bound to parameter markers (if any) in the SQL statement.
- A reference to a RuntimeDataList that describes the expected type and provides the buffers for values to be returned (if any) by the SQL statement.

The RemotePassthru and RemoteQuery classes are subclasses of the RemoteOperation class.

The RemoteOperation class is one of the operation classes for the Java API.

Package

com.ibm.db2.wrapper

Methods

The following table describes the methods of the RemoteOperation class. The methods are described in more detail after the table.

Table 134. Methods for the RemoteOperation class

Method	Description
getConnection	Retrieve the connection for the remote operation.
getExecDesc	Retrieve the execution descriptor for a remote operation.
getInputData	Retrieve the list of input values for the remote operation.
getOutputData	Retrieve the list of output data buffers from the remote operation.
getServer	Retrieve the data source server that owns the remote operation.
getWrapper	Retrieve the wrapper object.

Table 134. Methods for the RemoteOperation class (continued)

Method	Description
reportEof	Report an end-of-file condition during a fetch operation.

getConnection method

Purpose

Retrieve the connection for the remote operation.

Syntax

```
public final RemoteConnection getConnection()
```

Parameters

None.

Return value

The RemoteConnection instance.

Throws

None.

getExecDesc method

Purpose

Retrieve the execution descriptor for a remote operation.

Syntax

```
public final java.lang.Object getExecDesc()
                                    throws WrapperException,
                                           java.io.IOException,
                                           java.lang.ClassNotFoundException
```

Parameters

None.

Return value

The execution descriptor.

Throws

Any of the following exceptions.

- A WrapperException object if the retrieval of the execution descriptor object fails.
- java.io.IOException, if the streaming of the execution descriptor fails.
- java.lang.ClassNotFoundException, if the execution descriptor class cannot be found.

getInputData method

Purpose

Retrieve the list of input values for the remote operation.

Syntax

```
public final RuntimeDataList getInputData()
```

Parameters

None.

Return value

The list of input values.

Throws
None.

getOutputData method

Purpose
Retrieve the list of output data buffers for the remote operation.

Syntax

```
public final RuntimeDataList getOutputData()
```

Parameters
None.

Return value
The output data.

Throws
None.

getServer method

Purpose
Retrieve the data source server for the remote operation.

Syntax

```
public final FencedServer getServer()
```

Parameters
None.

Return value
The FencedServer instance where the remote operation runs.

Throws
None.

getWrapper method

Purpose
Retrieve the wrapper object.

Syntax

```
public final Wrapper getWrapper()
```

Parameters
None.

Return value
The wrapper object.

Throws
None.

reportEof method

Purpose
Report an end of file condition during a fetch operation.

Syntax

```
protected void reportEof()
                  throws WrapperException
```

Parameters
None.

Retrieve value
None.

Throws
A WrapperException object if the method fails.

Related reference:
- "FencedServer class (Java)" on page 71
- "RuntimeDataList class (Java)" on page 179
- "RemoteConnection class (Java)" on page 101
- "Wrapper class (Java)" on page 56
- "WrapperException class (Java)" on page 190

RemotePassthru class (Java)

This topic describes the RemotePassthru class and provides details for the constructors and methods.

Overview

The RemotePassthru class represents a pass-through session on a remote data source. An instance of your RemotePassthru subclass is created when the federated server invokes the createRemotePassthru method on an instance of the appropriate RemoteConnection subclass. If your data source does not support pass-through operations, do not implement a RemotePassthru subclass and do not override the default implementation of this method.

The public RemotePassthru class extends the RemoteOperation class.

The RemotePassthru class is one of the operation classes for the Java API.

Package
com.ibm.db2.wrapper

Constructors and methods

The following tables describe the constructor and methods of the RemotePassthru class. The constructor and methods are described in more detail after the tables.

Table 135. Constructors for the RemotePassthru class

Constructor	Description
RemotePassthru	Construct a new pass-through object for the specified connection.

Table 136. methods for the RemotePassthru class

method	Description
close	Close a pass-through cursor and allows the data source to clean up after pass-through statement processing.
describe	Describe the result set of a statement that is processed by a pass-through operation.
execute	Execute a statement that returns a code and does not return a result set by a pass-through operation.

Table 136. methods for the RemotePassthru class (continued)

method	Description
fetch	Fetch a row from a pass-through cursor by copying a single-result row into the output data buffer.
getSQLStatement	Retrieve the SQL statement for the remote pass-through operation.
open	Open a cursor for a pass-through operation.
prepare	Prepare a pass-through operation.
reportEof	Report an end-of-file condition during a fetch operation.

RemotePassthru constructor

Purpose

Construct a new object for the specified connection.

Syntax

```
protected RemotePassthru(RemoteConnection activeConnection,
                         long id)
```

Parameters

Table 137. Parameters for the RemotePassthru constructor

Name	Description
activeConnection	The connection through which the pass-through statement is submitted to the data source.
id	An integer value that represents the RemotePassthru object.

Throws

None.

close method

Purpose

Close a pass-through cursor and allows the data source to clean up after pass-through statement processing.

Syntax

```
protected void close()
          throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the close operation fails.

describe method

Purpose

Describe the result set of a statement that is processed by a pass-through

operation. Populates a RuntimeDataDescList class that describes the number and type of columns for each row of the result.

Syntax

```
protected void describe(RuntimeDataDescList dataDescriptionList)
              throws java.lang.Exception
```

Parameters

Table 138. Parameters for the describe method

Name	Description
dataDescriptionList	List of data descriptors for the operation.

Return value

None.

Throws

An Exception object if the method fails.

execute method

Purpose

Execute a statement that returns a code and does not return a result set by a pass-through operation.

Syntax

```
protected void execute()
             throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the execution process fails.

fetch method

Purpose

Fetch a row from a pass-through cursor by copying a single-result row into the output data buffer.

Syntax

```
protected void fetch()
           throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the fetch process fails.

getSQLStatement method

Purpose

Retrieve the SQL statement for the remote pass-through operation.

Syntax

```
public final java.lang.String getSQLStatement()
                                          throws WrapperException
```

Parameters

None.

Return value

A string value that represents the SQL statement.

Throws

A WrapperException object if the method fails.

open method

Purpose

Open a cursor for a pass-through operation and enable the data source to return the first result row for the query.

Syntax

```
protected void open()
          throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the open process fails.

prepare method

Purpose

Prepare a pass-through operation. Sends the pass-through string to the data source and determines the number and type of columns for each row of the result.

Syntax

```
protected void prepare(RuntimeDataDescList dataDescriptionList)
             throws java.lang.Exception
```

Parameters

Table 139. Parameters for the prepare method

Name	Description
dataDescriptionList	List of data descriptors for the operation.

Return value

None.

Throws

An Exception object if the prepare operation fails.

reportEof method

Purpose

Report an end-of-file condition during a fetch operation. The wrapper must call this method from the fetch() method when there are no more rows to retrieve.

Syntax

```
protected final void reportEof()
                     throws WrapperException
```

Overrides
The reportEof method in the RemoteOperation class.

Parameters
None.

Return value
None.

Throws
A WrapperException object if the method fails.

Inherited methods

The following methods are inherited from the RemoteOperation class and can be used with the RemotePassthru class:

- getConnection
- getExecDesc
- getInputData
- getOutputData
- getServer
- getWrapper

Related tasks:

- "Remote passthru class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "RuntimeDataDescList class (Java)" on page 188
- "RemoteConnection class (Java)" on page 101
- "RemoteOperation class (Java)" on page 108
- "WrapperException class (Java)" on page 190

RemoteQuery class (Java)

This topic describes the RemoteQuery class and provides details for the constructors and methods.

Overview

The RemoteQuery class represents SELECT statement operations on a remote data source. An instance of your RemoteQuery subclass is created when the federated server invokes the RemoteConnection.createRemoteQuery(long) method on an instance of the appropriate RemoteConnection subclass. The federated server destroys the RemoteQuery object after the application that submitted the query to the federated server closes its cursor over the result set of the query.

The public RemoteQuery class extends the RemoteOperation class.

The RemoteQuery class is one of the operation classes for the Java API.

Package
com.ibm.db2.wrapper

Class fields

The following table describes the valid class fields for the options that you can use with the RemoteQuery class.

Table 140. Class fields for the RemoteQuery class

Class field	Definition	Description
CLOSE_EOA	public static final byte CLOSE_EOA	Indicates that the wrapper is to finish necessary cleanup, and that the RemoteQuery object is to be destroyed after the close method returns. This constant is used only with the close method
CLOSE_EOS	public static final byte CLOSE_EOS	Indicates that the wrapper must leave the RemoteQuery object in a state where the reopen or reopenInputLob methods can be invoked. This constant is used only with the close method.
EOF	public static final byte EOF	Indicates that the RemoteQuery object finished fetching (retrieving) all of the rows of the result set. This constant is used only with the getStatus and setStatus methods.
FETCH_LOB_LAST	public static final int FETCH_LOB_LAST	Indicates that the last buffer of data for the current LOB column has been fetched (retrieved).
FETCH_LOB_MORE	public static final int FETCH_LOB_MORE	Indicates that more data are available for the current LOB column.
FETCH_LOB_NEXT	public static final int FETCH_LOB_NEXT	Indicates that a LOB column is to be the next fetched (retrieved) item.
FETCH_NON_LOB_NEXT	public static final int FETCH_NON_LOB_NEXT	Indicates that a non-LOB column is to be the next fetched (retrieved) item.
FETCH_ROW_DONE	public static final int FETCH_ROW_DONE	Indicates that all of the values for the current row have been fetched (retrieved).
OPEN	public static final byte OPEN	Indicates that the RemoteQuery object is in open state. This constant is used only with the getStatus and setStatus methods.

Table 140. Class fields for the RemoteQuery class (continued)

Class field	Definition	Description
OPEN_LOB_INIT	public static final int OPEN_LOB_INIT	Indicates the start of processing for the input LOB values.
OPEN_LOB_LAST	public static final int OPEN_LOB_LAST	Indicates that this is the last fragment for the current input LOB value.
OPEN_LOB_MORE	public static final int OPEN_LOB_MORE	Indicates that more fragments are available for the current input LOB value.
OPEN_NON_LOB_NEXT	public static final int OPEN_NON_LOB_NEXT	Indicates that only input non-LOB values are to be processed.
OPEN_LOB_NEW	public static final int OPEN_LOB_NEW	Indicates that the wrapper expects data fragments for a new input LOB value.
OPEN_ROW_DONE	public static final int OPEN_ROW_DONE	Indicates that all the input values have been processed.
READY	public static final byte READY	Indicates that the RemoteQuery object is ready to fetch (retrieve) the next row. This constant is used only with the getStatus and setStatus methods.
UNREADY	public static final byte UNREADY	Indicates that the RemoteQuery object is not yet ready to fetch the next row. This constant is used only with the getStatus and setStatus methods.

Constructor and methods

The following tables describe the constructor and methods of the RemoteQuery class. The constructor and methods are described in more detail after the tables.

Table 141. Constructors for the RemoteQuery class

Constructor	Description
RemoteQuery	Construct a new query object for the specified connection.

Table 142. Methods for the RemoteQuery class

Method	Description
close	Allow the wrapper and the data source to clean up after query statement processing.
fetch	Fetch (retrieve) a single result row from the remote data source query into the output data buffers.

Table 142. Methods for the RemoteQuery class (continued)

Method	Description
fetchLob	Retrieve a large object (LOB) fragment from the remote data source query.
getRowStatus	Retrieve the current row status.
getStatus	Retrieve the status of the query.
lobDataReady	Notify the federated server that a fragment of data from a LOB column is fetched (retrieved).
open	Allow the wrapper to prepare the data source to return the first result row for the query.
openInputLob	Allow the wrapper to prepare the remote data source to return the first result row for the query that contains input LOB parameters.
reopen	Reset a previously opened result stream and prepare the data source to return more results. These results might be based on different parameter bindings.
reopenInputLob	Reset a previously opened result stream and prepares the data source to return more result sets. These results might be based on different parameter bindings for queries with input LOB parameters.
reportEof	Report an end-of-file condition during a fetch (retrieve) operation.
setLobNext	Notify the federated server when a current fetched (retrieved) row contains LOBs and that the fetchLob method is to be invoked.
setRowStatus	Set the current row status.
setStatus	Set the query status.

RemoteQuery constructor

Purpose

Construct a new query object for the specified connection.

Syntax

```
protected RemoteQuery(RemoteConnection activeConnection,
                      long id)
```

Parameters

Table 143. Parameters for the RemoteQuery constructor

Name	Description
activeConnection	Connection through which the query runs.
id	Integer value that represents the RemoteQuery object.

Throws

None.

close method

Purpose

Allow the wrapper and the data source to clean up after query statement processing. The wrapper must implement this method in the wrapper-specific RemoteQuery subclass.

Usage A single SQL statement that is submitted to the federated server can result in multiple query requests to a wrapper. So that wrappers can more easily optimize the translation and submission of queries to the data source, the close method receives a status flag that indicates whether the close represents an end-of-query (CLOSE_EOS) or an end-of-statement (CLOSE_EOA) status. If the close method is called with an end-of-query status, the RemoteQuery object can execute the reopen method successfully. If the close method is called with an end-of-statement status, the federated server destroys the RemoteQuery object after the close method completes.

Syntax

```
protected void close(short status)
          throws java.lang.Exception
```

Parameters

Table 144. Parameters for the close method

Name	Description
status	Status of the operation.

Return value

None.

Throws

An Exception object if the close operation fails.

fetch method

Purpose

Fetch (retrieve) a single result row from the remote data source query into the output data buffers. The wrapper must implement this method in the wrapper-specific RemoteQuery subclass.

Syntax

```
protected void fetch()
          throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the fetch (retrieve) process fails.

fetchLob method

Purpose

Retrieve a large object (LOB) fragment from the remote data source query. The wrapper can implement this method in the wrapper-specific RemoteQuery subclass if the wrapper transfers LOBs.

Usage The fetchLob method includes a function argument that specifies the

maximum size of the fetched (retrieved) data fragment. The fetchLob method must call the lobDataReady method to inform the federated server of the fetch status.

Syntax

```
protected java.lang.Object fetchLob(int bufferSize,
                                    int bytesWritten)
                             throws java.lang.Exception
```

Parameters

Table 145. Parameters for the fetchLob method

Name	Description
bufferSize	Maximum size of the data fragment to be fetched (retrieved).
bytesWritten	Number of bytes already fetched (retrieved) for the current LOB.

Return value

A data fragment. This data fragment is a string for CLOB data and is a byte array for BLOB data.

Throws

A WrapperException object if the fetchLob method fails.

getRowStatus method

Purpose

Retrieve the current row status.

Syntax

```
protected final int getRowStatus()
```

Parameters

None.

Return value

The current row status.

Throws

None.

getStatus method

Purpose

Retrieve the status of the query.

Syntax

```
protected final byte getStatus()
```

Parameters

None.

Return value

The query status.

Throws

None.

lobDataReady method

Purpose

Notify the federated server that a fragment of data from a LOB column is fetched (retrieved). Call this method from the fetchLob method.

Syntax

```
protected final void lobDataReady(int columnNumber,
                                  int bytesReady,
                                  int status,
                                  int intent)
                           throws WrapperException
```

Parameters

Table 146. Parameters for the lobDataReady method

Name	Description
columnNumber	Column index that this data fragment belongs to.
bytesReady	Length of the fragment in bytes.
status	Status of the LOB operation. The status value can be FETCH_LOB_MORE or FETCH_LOB_LAST.
intent	Flag that indicates if there are any more LOB or non-LOB columns to fetch (retrieve), or if the complete row was fetched.

Return value

None.

Throws

A WrapperException object if the fetch (retrieve) process fails

open method

Purpose

Allow the wrapper to prepare the data source to return the first result row for the query. The wrapper must implement this method in the wrapper-specific RemoteQuery subclass.

Syntax

```
protected void open()
          throws java.lang.Exception
```

Parameters

None.

Return value

None.

Throws

An Exception object if the open process fails

openInputLob method

Purpose

Allow the wrapper to prepare the data source to return the first result row for the query that contains LOBs. If the wrapper supports input LOB parameters, the wrapper must implement this method in the wrapper-specific RemoteQuery subclass.

Usage The federated server calls this method if input LOB host variables are found. Initially, the openInputLob method is called with an empty buffer and the columnNumber parameter set to -1. The wrapper must return the index of the host variable that receives the next LOB fragment, then call the setRowStatus method to indicate that there are input LOB parameters. When the wrapper indicates that there are more LOB input parameters to process, the federated server calls the openInputLob method with another LOB fragment. The LOB input value total size, in bytes, is given as the matSize parameter. The size of the current LOB fragment, in bytes, is given as the xferBytes parameter. The wrapper must use these parameter values to either advance to the next input variable, or to signal that the entire input values have been read.

Syntax

```
protected int openInputLob(int columnNumber,
                           int matSize,
                           int xferBytes,
                           java.lang.Object buffer)
                    throws java.lang.Exception
```

Parameters

Table 147. Parameters for the lobDataReady method

Name	Description
columnNumber	Index of the input host variable that receives the current LOB fragment.
matSize	Materialized size of the input data, in bytes,
xferBytes	Size of the current data fragment, in bytes.
buffer	Current LOB fragment object. The LOB fragment object is a string for CLOB variables, and is a byte array for BLOB variables.

Return value
The index of the input host variable that receives the next LOB fragment.

Throws
An Exception object if the openInputLob process fails

reopen method

Purpose
Reset a previously opened result stream and prepare the data source to return more results. These results might be based on different parameter bindings. The wrapper must implement this method in the wrapper-specific RemoteQuery subclass. The reopen method is not called unless the query was previously closed with an end-of-query status.

Syntax

```
protected void reopen(short action)
              throws java.lang.Exception
```

Parameters

Table 148. Parameters for the reopen method

Name	Description
action	Unused.

Return value
None.

Throws
An Exception object if the reopen process fails.

reopenInputLob method

Purpose
Reset a previously opened result stream and prepare the data source to return more result sets. These results might be based on different parameter bindings for queries with input LOB parameters. If the wrapper supports input LOB parameters, the wrapper must implement this method in the wrapper-specific RemoteQuery subclass.

Usage This method is not called unless the query was previously closed with an end-of-query status. The federated server calls this method if input LOB host variables are found. Initially, the reopenInputLob method is called with an empty buffer and the columnNumber parameter set to -1. The wrapper must return the index of the host variable that receives the next LOB fragment, then call the setRowStatus method to indicate that there are input LOB parameters. When the wrapper indicates that there are more LOB input parameters to process, the federated server calls the reopenInputLob method with another LOB fragment. The LOB input value total size, in bytes, is given as the matSize parameter. The size of the current LOB fragment, in bytes, is given as the xferBytes parameter. The wrapper must use these parameter values to either advance to the next input variable, or to signal that the entire input values have been read.

Syntax

```
protected int reopenInputLob(short action,
                             int columnNumber,
                             int matSize,
                             int xferBytes,
                             java.lang.Object buffer)
                      throws java.lang.Exception
```

Parameters

Table 149. Parameters for the reopenInputLob method

Name	Description
action	Unused.
columnNumber	Index of the input host variable that receives the current LOB fragment.
matSize	Materialized size of the input data, in bytes.
xferBytes	Size of the current data fragment, in bytes.
buffer	Current LOB fragment object. The LOB fragment object is a string for CLOB variables, and is a byte array for BLOB variables.

Return value
The index of the input host variable that receives the next LOB fragment.

Throws
An Exception object if the reopenInputLob process fails

reportEof method

Purpose

Report an end-of-file condition during a fetch (retrieve) operation. The wrapper-specific RemoteQuery subclass invokes this method during the fetch operation to indicate that the last row was fetched.

Syntax

```
protected final void reportEof()
                     throws WrapperException
```

Overrides

The reportEof method in the RemoteOperation class.

Parameters

None.

Return value

None.

Throws

A WrapperException object if the method fails.

setLobNext method

Purpose

Notify the federated server when a current fetched (retrieved) row contains LOBs, and that the fetchLob method is to be invoked.

Usage Call this method from the fetch() method if there is LOB data to transfer. Do *not* call the setLobNext method from inside the fetchLob method. After the fetch method returns and setLobNext is called, the federated server calls fetchLob to retrieve the LOB column values.

Syntax

```
protected final void setLobNext()
```

Parameters

None.

Return value

None.

Throws

None.

setRowStatus method

Purpose

Set the current row status.

Syntax

```
pprotected final void setRowStatus(int status)
```

Parameters

Table 150. Parameters for the setStatus method

Name	Description
status	New row status to be set.

Return value

None.

Throws
None.

setStatus method

Purpose
Set the query status.

Syntax

```
protected final void setStatus(byte status)
```

Parameters

Table 151. Parameters for the setStatus method

Name	Description
status	New query status to be set.

Return value
None.

Throws
None.

Inherited methods

The following methods are inherited from the RemoteOperation class and can be used with the RemoteQuery class:

- getConnection
- getExecDesc
- getInputData
- getOutputData
- getServer
- getWrapper

Related reference:

- "Remote query class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*
- "RemoteConnection class (Java)" on page 101
- "RemoteOperation class (Java)" on page 108
- "WrapperException class (Java)" on page 190

Planning classes for the Java API

The following table describes each planning class for the Java API.

Table 152. Request classes

Class name	Description
ParsedQueryFragment	The class that describes a query fragment.
Request	The class that encapsulates a query fragment that is analyzed and processed by the wrapper.
Reply	The class that represents a portion of a request that the wrapper processes.
RequestExp	The class that represents a node in an expression tree.

Table 152. Request classes (continued)

Class name	Description
RequestExpType	The class that describes the type of node in an expression tree.
RequestConstant	The class that describes the constants that are used in a query expression during query planning.
Quantifier	The class that encapsulates information for a quantifier of a request.
PredicateList	The class that describes two lists of predicates (a list of predicates for which the selectivity is solicited and a list of previously applied predicates) to estimate the selectivity factor.

Related reference:

- "PredicateList class (Java)" on page 154
- "ParsedQueryFragment class (Java)" on page 126
- "Quantifier class (Java)" on page 152
- "Reply class (Java)" on page 131
- "RequestConstant class (Java)" on page 147
- "Request class (Java)" on page 130
- "RequestExpType class (Java)" on page 144
- "RequestExp class (Java)" on page 139

ParsedQueryFragment class (Java)

This topic describes the ParsedQueryFragment class and provides details for the methods.

This class does not contain constructors.

Overview

The ParsedQueryFragment class represents the base class of both the Request class and the Reply class. The ParsedQueryFragment class describes a query fragment. The query fragment consists of SELECT, FROM, WHERE, and ORDER BY clauses. Each clause is represented by an array of objects. The methods of this class return the size of the arrays, the entries, and the associated data structures.

The Reply class and the Request class are subclasses of the ParsedQueryFragment class.

The ParsedQueryFragment class is one of the planning classes for the Java API.

Package

com.ibm.db2.wrapper

Methods

The following table describes the methods of the ParsedQueryFragment class. The methods are described in more detail after the table.

Table 153. Methods for the ParsedQueryFragment class

Method	Description
getDistinct	Test the DISTINCT indicator (SELECT DISTINCT clause).
getHeadExp	Retrieve the expression at the specified position in the SELECT clause.
getNickname	Retrieve the nickname that corresponds to the specified position in the FROM clause.
getNumberOfHeadExp	Retrieve the number of elements in the SELECT clause.
getNumberOfPredicates	Retrieve the number of elements (predicates) in the WHERE clause.
getNumberOfQuantifiers	Retrieve the number of elements (quantifiers) in the FROM clause.
getPredicate	Retrieve the predicate expression at the specified position in the WHERE clause.
getQuantifier	Retrieve a quantifier at the specified position in the FROM clause.
getQuantiferByHandle	Retrieve the quantifier with the specified handle in the FROM clause.

getDistinct method

Purpose

Test the DISTINCT indicator (SELECT DISTINCT clause).

Syntax

```
public final boolean getDistinct()
```

Parameters

None.

Return value

The DISTINCT indicator.

Throws

None.

getHeadExp method

Purpose

Retrieve the expression at the specified position in the SELECT clause.

Syntax

```
public final RequestExp getHeadExp(int position)
```

Parameters

Table 154. Parameters for the getHeadExp method

Name	Description
position	Position of the head expression in the SELECT clause. The first head expression is at position 1.

Return value
The head expression at the specified position.

Throws
None.

getNickname method

Purpose
Retrieve the nickname that corresponds to the specified position in the FROM clause.

Syntax

```
public final UnfencedGenericNickname getNickname(int position)
```

Parameters

Table 155. Parameters for the getNickname method

Name	Description
position	Position of the quantifier in the FROM clause. The first quantifier is at position 1.

Return value
The nickname object at the specified position.

Throws
None.

getNumberOfHeadExp method

Purpose
Retrieve the number of elements in the SELECT clause.

Syntax

```
public final int getNumberOfHeadExp()
```

Parameters
None.

Return value
The number of head expressions.

Throws
None.

getNumberOfPredicates method

Purpose
Retrieve the number of elements (predicates) in the WHERE clause.

Syntax

```
public final int getNumberOfPredicates()
```

Parameters
None.

Return value
The number of predicates.

Throws
None.

getNumberOfQuantifiers method

Purpose
Retrieve the number of elements (quantifiers) in the FROM clause.

Syntax

```
public final int getNumberOfQuantifiers()
```

Parameters
None.

Return value
The number of quantifiers.

Throws
None.

getPredicate method

Purpose
Retrieve the predicate expression at the specified position in the WHERE clause.

Syntax

```
public final RequestExp getPredicate(int position)
```

Parameters

Table 156. Parameters for the getPredicate method

Name	Description
position	The position of the predicate in the WHERE clause. The first predicate is at position 1..

Return value
The predicate expression at the specified position.

Throws
None.

getQuantifier method

Purpose
Retrieve a quantifier at the specified position in the FROM clause.

Syntax

```
public final Quantifier getQuantifier(int position)
```

Parameters

Table 157. Parameters for the getQuantifier method

Name	Description
position	The quantifier position in the FROM clause. The first quantifier is at position 1.

Return value
The quantifier at the specified position.

Throws
None.

getQuantiferByHandle method

Purpose
Retrieve the quantifier with the specified handle in the FROM clause.

Syntax

```
public final Quantifier getQuantiferByHandle(int quantifierHandle)
```

Parameters

Table 158. Parameters for the getQuantiferByHandle method

Name	Description
quantifierHandle	The quantifier handle.

Return value
The quantifier object for the specified handle.

Throws
None.

Related reference:
- "Quantifier class (Java)" on page 152
- "Reply class (Java)" on page 131
- "Request class (Java)" on page 130
- "RequestExp class (Java)" on page 139
- "UnfencedGenericNickname class (Java)" on page 99

Request class (Java)

This topic describes the Request class.

Overview

The Request class encapsulates a query fragment that is analyzed and processed by the wrapper. The federated server provides instances of this class to the UnfencedGenericServer.planRequest method, which returns the appropriate Reply objects.

The public Request class extends the ParsedQueryFragment class.

The Request class is one of the planning classes for the Java API.

Package
com.ibm.db2.wrapper

Inherited methods

The following methods are inherited from the ParsedQueryFragment class and can be used with the Request class:

- getDistinct
- getHeadExp
- getNickname
- getNumberOfHeadExp
- getNumberOfPredicates
- getNumberOfQuantifiers
- getPredicate
- getQuantifier
- getQuantiferByHandle

Related tasks:

- "Request class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "ParsedQueryFragment class (Java)" on page 126
- "Reply class (Java)" on page 131
- "UnfencedGenericServer class (Java)" on page 78

Reply class (Java)

This topic describes the Reply class provides details for the constructor and methods.

Overview

The Reply class represents the portion of the request that the wrapper processes. In addition to the actions of the ParsedQueryFragment class, the Reply class adds methods and data members to initiate the following actions:

- Populate the reply by adding entries to the clauses.
- Store the execution descriptor structure.
- Chain the replies when the wrapper returns more than one plan for the same request.
- Store order information. The optimizer might use returned data in different ordering sequences to construct a more optimal plan.

The public Request class extends the ParsedQueryFragment class.

The Reply class is one of the planning classes for the Java API.

Usage Advanced customization is available with the Reply class by using an execution cost (estimated execution time) for the reply query fragment. The wrapper writer can override the default implementation of the costing methods with methods that more precisely describe the execution model of the source. Overriding the default costing is dependent on the default selectivity estimation for the predicates in the query fragment. Selectivity estimates are obtained by calling the UnfencedGenericServer.getSelectivity method that can also be overloaded by the wrapper writer.

Package

com.ibm.db2.wrapper

Constants

The following table describes the valid constants that you can use with the Reply class.

Table 159. Constants for the Reply class

Constant	Definition	Description
ASC	static final int ASC	Constant that indicates the ascending order for an order entry.
DESC	public static final int DESC	Constant that indicates the descending order for an order entry.

Constructor and methods

The following tables describe the constructor and the methods of the Reply class. The constructor and methods are described in more detail after the tables.

Table 160. Constructors for the Reply class

Constructor	Description
Reply	Instantiate a Reply for the specified data source (Server) and Request.

Table 161. Methods for the Reply class

Method	Description
addHeadExp	Add a head expression in the SELECT clause of the query.
addPredicate	Add a predicate in the WHERE clause of the query.
addQuantifier	Add a quantifier in the FROM clause of the query.
addOrderEntry	Add an order entry in the ORDER BY clause of the query.
cardinality	Retrieve the cardinality of the result when running the query fragment that is represented by the reply.
firstTupleCost	Retrieve the cost that is required to obtain the first result tuple for the query fragment that is represented by the reply.
getExecDesc	Retrieve the execution descriptor object.
getNumberOfOrderEntries	Retrieve the number of order entries.
getOrderEntry	Retrieve the order entry at the specified position in the ORDER BY clause.
getServer	Retrieve the data source server that this reply belongs to.
nextReply	Retrieve the next reply in the chain.
reExecCost	Retrieve the cost needed to rerun the query fragment that is represented by the reply.

Table 161. Methods for the Reply class (continued)

Method	Description
setDistinct	Set the DISTINCT indicator that is specified in the SELECT DISTINCT statement.
setExecDesc	Set the execution descriptor object.
setNextReply	Specify the next reply in the chain.
totalCost	Retrieve the total execution cost that is needed to run the query fragment that is represented by the reply.

Reply constructor

Purpose

Instantiate a Reply for the specified data source (Server) and Request.

Syntax

```
public Reply(Request request,
             UnfencedGenericServer server)
```

Parameters

Table 162. Parameters for the Reply constructor

Name	Description
request	The Request object for which the Reply is generated. The Request object encapsulates a query fragment that is to be analyzed and processed by the wrapper.
server	The Server object that is processing the Request and generates the Reply.

addHeadExp method

Purpose

Add a head expression in the SELECT clause of the query.

Syntax

```
public final void addHeadExp(RequestExp headExp)
```

Parameters

Table 163. Parameters for the addHeadExp method

Name	Description
headExp	Head expression.

Return value

None.

Throws

None.

addPredicate method

Purpose

Add a predicate in the WHERE clause of the query.

Syntax

```
public final void addPredicate(RequestExp predicate)
```

Parameters

Table 164. Parameters for the addPredicate method

Name	Description
predicate	Predicate to add.

Return value

None.

Throws

None.

addQuantifier method

Purpose

Add a quantifier in the FROM clause of the query.

Syntax

```
public final void addQuantifier(Quantifier quantifier)
```

Parameters

Table 165. Parameters for the addQuantifier method

Name	Description
quantifier	Quantifier to add.

Return value

None.

Throws

None.

addOrderEntry method

Purpose

Add an order entry in the ORDER BY clause of the query. An order entry consists of the head expression position and the direction (ASC or DESC).

Syntax

```
public final void addOrderEntry(int position,
                                int direction)
                         throws WrapperException
```

Parameters

Table 166. Parameters for the addOrderEntry method

Name	Description
position	Position of the head expression in the SELECT clause.
direction	Direction, which either is ascending (ASC) order or descending (DESC) order.

Return value
None.

Throws
A WrapperException object if the direction is not valid.

cardinality method

Purpose
Retrieve the cardinality of the result when running the query fragment that is represented by the reply.

Syntax

```
public double cardinality()
```

Parameters
None.

Return value
The cardinality.

Throws
None.

firstTupleCost method

Purpose
Retrieve the cost that is required to obtain the first result tuple for the query fragment that is represented by the reply.

Syntax

```
public double firstTupleCost()
```

Parameters
None.

Return value
The first tuple cost.

Throws
None.

getExecDesc method

Purpose
Retrieve the execution descriptor object.

Syntax

```
public final java.io.Serializable getExecDesc()
```

Parameters
None.

Return value
The execution descriptor.

Throws
None.

getNumberOfOrderEntries method

Purpose

Retrieve the number of order entries.

Syntax

```
public final int getNumberOfOrderEntries()
```

Parameters

None.

Return value

The number of order entries.

Throws

None.

getOrderEntry method

Purpose

Retrieve the order entry at the specified position in the ORDER BY clause. An order entry consists of the head expression position and the direction (ASC or DESC).

Syntax

```
public final int[] getOrderEntry(int position)
                            throws WrapperException
```

Parameters

Table 167. Parameters for the getOrderEntry method

Name	Description
position	The position of the order entry in the ORDER BY clause. The first entry is at position 1.

Return value

An array of two integer values. The two integer values are the head expression in the SELECT clause and the order type, either ASC (ascending) or DESC (descending).

Throws

A WrapperException object if the requested position is not valid.

getServer method

Purpose

Retrieve the data source server that this reply belongs to.

Syntax

```
public final UnfencedGenericServer getServer()
```

Parameters

None.

Return value

The data source server of the reply.

Throws

None.

nextReply method

Purpose

Retrieve the next reply in the chain.

Syntax

```
public final Reply nextReply()
```

Parameters

None.

Return value

The next reply in the chain, or null if there are no more replies.

Throws

None.

reExecCost method

Purpose

Retrieve the cost needed to rerun the query fragment that is represented by the reply.

Syntax

```
public double reExecCost()
```

Parameters

None.

Return value

The re-execution cost.

Throws

None.

setDistinct method

Purpose

Set the DISTINCT indicator that is specified in the SELECT DISTINCT clause.

Syntax

```
public final void setDistinct(boolean distinct)
```

Parameters

Table 168. Parameters for the setDistinct method

Name	Description
distinct	The DISTINCT indicator.

Retrieve value

None.

Throws

None.

setExecDesc method

Purpose

Set the execution descriptor object.

Syntax

```
public final void setExecDesc(java.io.Serializable execDesc)
```

Parameters

Table 169. Parameters for the setExecDesc method

Name	Description
execDesc	Execution descriptor.

Return value

None.

Throws

None.

setNextReply method

Purpose

Set the next reply in the chain. Chain the replies when the wrapper returns more than one reply for a request.

Syntax

```
public final void setNextReply(Reply next)
```

Parameters

Table 170. Parameters for the setNextReply method

Name	Description
next	The Reply object to be added in the chain.

Return value

None.

Throws

None.

totalCost method

Purpose

Retrieve the total execution cost that is needed to run the query fragment that is represented by the reply.

Syntax

```
public double totalCost()
```

Parameters

Return value

The total cost.

Throws

None.

Inherited methods

The following methods are inherited from the ParsedQueryFragment class and can be used with the Reply class:

- getDistinct
- getHeadExp
- getNickname
- getNumberOfHeadExp
- getNumberOfPredicates
- getNumberOfQuantifiers
- getPredicate
- getQuantifier
- getQuantifierByHandle

Related tasks:

- "Reply class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "ParsedQueryFragment class (Java)" on page 126
- "Request class (Java)" on page 130
- "UnfencedGenericServer class (Java)" on page 78

RequestExp class (Java)

This topic describes the RequestExp class and provides details for each method.

This class does not contain constructors.

Overview

The RequestExp class represents a node in an expression tree. This node can be a column reference, a constant value, an unbound parameter, or an operator. An unbound parameter is similar to a constant. However, the unbound parameter value is unknown until the runtime phase when the federated server passes the value to the wrapper.

The RequestExp class is one of the planning classes for the Java API.

Usage A RequestExp object is never instantiated by the wrapper. The federated server creates these objects and passes them to the wrapper during query planning.

Package
com.ibm.db2.wrapper

Constants

The following table describes the valid constants for the options that you can use with the RequestExp class.

Table 171. Constants for the RequestExp class

Constant	Definition	Description
BADKIND	public static final int BADKIND	Indicates that the expression is unknown.

Table 171. Constants for the RequestExp class (continued)

Constant	Definition	Description
COLUMN	public static final int COLUMN	Indicates that the expression is a column.
CONSTANT	public static final int CONSTANT	Indicates that the expression is a constant.
OPERATOR	public static final int OPERATOR	Indicates that the expression is an operator.
UNBOUND	public static final int UNBOUND	Indicates that the expression is an unbound parameter.

Methods

The following table describes the methods of the RequestExp class. The methods are described in more detail after the table.

Table 172. Methods for the RequestExp class

Method	Description
getColumnName	Retrieve the column name.
getDataType	Retrieve the object that describes the data type of the expression.
getFirstChild	Retrieve the first child of the operator.
getHandle	Retrieve the expression handle.
getKind	Retrieve the kind of expression.
getNextChild	Retrieve the sibling of the expression.
getNumberofChildren	Retrieve the number of children for the operator.
getParent	Retrieve the parent node of the expression.
getQuantifier	Retrieve the quantifier to which this column belongs.
getSignature	Retrieve the signature of the operator.
getToken	Retrieve the token of the operator.
getValue	Retrieve the RequestConstant object that represents the value of the constant.

getColumnName method

Purpose

Retrieve the column name (column expression node).

Syntax

```
public java.lang.String getColumnName()
          throws WrapperException
```

Parameters

None.

Return value

The column name.

Throws

A WrapperException object if method is called for non-column expression nodes.

getDataType method

Purpose

Retrieve the object that describes the data type of the expression.

Syntax

```
public RequestExpType getDataType()
```

Parameters

None.

Return value

The expression type.

Throws

None.

getFirstChild method

Purpose

Retrieve the first child of the operator (operator expression node).

Syntax

```
public RequestExp getFirstChild()
           throws WrapperException
```

Parameters

None.

Return value

The first child.

Throws

A WrapperException object if method is called for non-operator expression nodes.

getHandle method

Purpose

Retrieve the expression handle.

Syntax

```
public int getHandle()
```

Parameters

None.

Return value

The handle.

Throws

None.

getKind method

Purpose

Retrieve the kind of expression.

Syntax

```
public int getKind()
```

Parameters

None.

Return value

The kind of expression.

Throws

None.

getNextChild method

Purpose

Retrieve the sibling of the expression.

Syntax

```
public RequestExp getNextChild()
```

Parameters

None.

Return value

The next sibling or null if the expression does not have a sibling.

Throws

None.

getNumberOfChildren method

Purpose

Retrieve the number of children for the operator (operator expression node).

Syntax

```
public int getNumberOfChildren()
           throws WrapperException
```

Parameters

None.

Return value

The number of children.

Throws

A WrapperException object if method is called for non-operator expression nodes.

getParent method

Purpose

Retrieve the parent node of the expression.

Syntax

```
public RequestExp getParent()
```

Parameters
None.

Return value
The parent expression.

Throws
None.

getQuantifier method

Purpose
Retrieve the column expression node (the quantifier) to which this column belongs.

Syntax

```
public Quantifier getQuantifier()
        throws WrapperException
```

Parameters
None.

Return value
The quantifier.

Throws
A WrapperException object if method is called for non-column expression nodes.

getSignature method

Purpose
Retrieve the signature of the operator (operator expression node).

Syntax

```
public java.lang.String getSignature()
        throws WrapperException
```

Parameters
None.

Return value
The signature.

Throws
A WrapperException object if method is called for non-operator expression nodes.

getToken method

Purpose
Retrieve the token of the operator (operator expression node).

Syntax

```
public java.lang.String getToken()
        throws WrapperException
```

Parameters
None.

Return value
The token of the operator.

Throws
A WrapperException object if method is called for non-operator expression nodes.

getValue method

Purpose
Retrieve the RequestConstant object that represents the value of the constant (constant expression node).

Syntax

```
public RequestConstant getValue()
         throws WrapperException
```

Parameters
None.

Return value
The value of the constant as a RequestConstant instance.

Throws
A WrapperException object if method is called for non-constant expression nodes.

Related reference:

- "Request expression class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*
- "RequestExpType class (Java)" on page 144

RequestExpType class (Java)

This topic describes the RequestExpType class and provides details for each method.

This class does not contain constructors.

Overview

The RequestExpType class describes the type of node in an expression tree. This node can be a column reference, a constant value, an unbound parameter, or an operator.

The RequestExpType class is one of the planning classes for the Java API.

Usage The wrapper never instantiates a RequestExpType object. The federated server creates these objects and passes them to the wrapper during query planning.

Package
com.ibm.db2.wrapper

Methods

The following table describes the methods of the RequestExpType class. The methods are described in more detail after the table.

Table 173. Methods for the RequestExpType class

Method	Description
getCodepage	Retrieve the code page for character data types.
getDataType	Retrieve the data type.
getForBitData	Retrieve the FOR BIT DATA flag, which indicates whether the data is in binary format.
getMaximumLength	Retrieve the maximum length for the data type.
getName	Retrieve the name of the data type.
getNullIndicator	Retrieve the null indicator value.
getPrecision	Retrieve the precision for numeric data types.
getScale	Retrieve the scale for numeric data types.

getCodepage function

Purpose

Retrieve the code page for character data types.

Syntax

```
public short getCodepage()
```

Parameters

None.

Return value

The code page.

Throws

None.

getDataType function

Purpose

Retrieve the data type.

Syntax

```
public short getDataType()
```

Parameters

None.

Return value

The data type.

Throws

None.

getForBitData function

Purpose

Retrieve the FOR BIT DATA flag, which indicates whether the data is in binary format.

Syntax

```
public boolean getForBitData()
```

Parameters

None.

Return value

The FOR BIT DATA flag.

Throws

None.

getMaximumLength function

Purpose

Retrieve the maximum length for the data type.

Syntax

```
public int getMaximumLength()
```

Parameters

None.

Return value

The maximum length for the data type.

Throws

None.

getName function

Purpose

Retrieve the name of the data type.

Syntax

```
public java.lang.String getName()
```

Parameters

None.

Return value

The name.

Throws

None.

getNullIndicator function

Purpose

Retrieve the null indicator value.

Syntax

```
public short getNullIndicator()
```

Parameters
None.

Return value
The null indicator.

Throws
None.

getPrecision function

Purpose
Retrieve the precision for numeric data types.

Syntax

```
public byte getPrecision()
```

Parameters
None.

Return value
The precision.

Throws
None.

getScale function

Purpose
Retrieve the scale for numeric data types.

Syntax

```
public byte getScale()
```

Parameters
None.

Return value
The scale.

Throws
None.

Related reference:

- "Request expression type class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*
- "RequestExp class (Java)" on page 139

RequestConstant class (Java)

This topic describes the RequestConstant class and provides details for the methods.

This class does not contain constructors.

Overview

The RequestConstant class describes the constants that are used in a query expression during query planning. The RequestExp.getValue() method returns an instance of the RequestConstant class.

The public RequestConstant class extends the Data class.

The RequestConstant class is one of the planning classes for the Java API.

Package

com.ibm.db2.wrapper

Constants

The following constants are inherited from the Data class and can be used with the RequestConstant class:

Table 174. Constants for the RequestConstant class

Constant	Definition	Description
BLOB	public static final short BLOB	Constant that indicates a BLOB data type.
CHAR	public static final short CHAR	Constant that indicates a CHAR data type.
CLOB	public static final short CLOB	Constant that indicates a CLOB data type.
DATE	public static final short DATE	Constant that indicates a DATE data type.
DECIMAL	public static final short DECIMAL	Constant that indicates a DECIMAL or NUMERIC data type.
DOUBLE	public static final short DOUBLE	Constant that indicates a DOUBLE data type.
FLOAT	public static final short FLOAT	Constant that indicates a FLOAT data type.
INT	public static final short INT	Constant that indicates an INTEGER (INT) data type.
LONG	public static final short LONG	Constant that indicates a LONG data type.
NONE	public static final short NONE	Constant that indicates an unknown data type.
SHORT	public static final short SHORT	Constant that indicates a SHORT data type.
TIME	public static final short TIME	Constant that indicates a TIME data type.
TIMESTAMP	public static final short TIMESTAMP	Constant that indicates a TIMESTAMP data type.
VARCHAR	public static final short VARCHAR	Constant that indicates a VARCHAR data type.

Methods

The following table describes the methods of the RequestConstant class. The methods are described in more detail after the table.

Table 175. Methods for the RequestConstant class

Method	Description
getCodepage	Retrieve the code page for character data types.
getData	Retrieve the data buffer in an internal format.
getDataType	Retrieve the data type.
getForBitData	Retrieve the FOR BIT DATA flag.
getMaximumLength	Retrieve the maximum length for the data type.
getName	Retrieve the name of the constant, if defined.
getNullIndicator	Retrieve the null indicator.
getPrecision	Retrieve the precision for numeric data types.
getScale	Retrieve the scale for numeric data types.
isDataNull	Indicate if the constant is null.

getCodepage method

Purpose

Retrieve the code page for character data types.

Syntax

```
public short getCodepage()
```

Parameters

None.

Return value

The code page.

Throws

None.

getData method

Purpose

Retrieve the data buffer in an internal format.

Syntax

```
protected byte[] getData()
```

Overrides

The getData method in the Data class.

Parameters

None.

Return value

The data.

Throws

None.

getDataType method

Purpose

Retrieve the data type.

Syntax

```
public short getDataType()
```

Overrides
The getDataType method in the Data class.

Parameters
None.

Return value
The data type.

Throws
None.

getForBitData method

Purpose
Retrieve the FOR BIT DATA flag.

Syntax

```
public boolean getForBitData()
```

Overrides
The getForBitData method in the Data class.

Parameters
None.

Return value
The FOR BIT DATA flag.

Throws
None.

getMaximumLength method

Purpose
Retrieve the maximum length for the data type.

Syntax

```
public int getMaximumLength()
```

Parameters
None.

Return value
The maximum length.

Throws
None.

getName method

Purpose
Retrieve the name of the constant, if defined.

Syntax

```
public java.lang.String getName()
```

Parameters
None.

Return value
The name.

Throws
None.

getNullIndicator method

Purpose
Retrieve the null indicator.

Syntax

```
public short getNullIndicator()
```

Parameters
None.

Return value
The null indicator.

Throws
None.

getPrecision method

Purpose
Retrieve the precision for numeric data types.

Syntax

```
public byte getPrecision()
```

Parameters
None.

Return value
The precision.

Throws
None.

getScale method

Purpose
Retrieve the scale for numeric data types.

Syntax

```
public byte getScale()
```

Parameters
None.

Return value
The scale.

Throws
None.

isDataNull method

Purpose

Indicate if the constant is null.

Syntax

```
public boolean isDataNull()
```

Parameters

None.

Return value

A value of true if the constant is null. Otherwise, the value is false.

Throws

None.

Inherited methods

The following methods are inherited from the Data class and can be used with the RequestConstant class:

- getBigDecimal
- getByte
- getDate
- getDouble
- getFloat
- getInt
- getLong
- getObject
- getShort
- getString
- getTime
- getTimestamp

Related reference:

- "Request constant class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*
- "Data class (Java)" on page 156
- "RequestExp class (Java)" on page 139

Quantifier class (Java)

This topic describes the Quantifier class and provides details for each method.

This class does not contain constructors.

Overview

The Quantifier class encapsulates information for a quantifier of a Request.

The Quantifier class is one of the planning classes for the Java API.

Usage This class is instantiated by the federated server to contain information for a Request object during query planning. The planRequest method in the

wrapper-specific unfenced server subclass examines the quantifier information. If the wrapper can process this quantifier information, the planRequest method adds the quantifier information to the Reply objects.

Package

com.ibm.db2.wrapper

Methods

The following table describes the methods of the Quantifier class. The methods are described in more detail after the table.

Table 176. methods for the Quantifier class

Method	Description
getHandle	Retrieve the handle of the quantifier.
getNickname	Retrieve the nickname that is referenced by this quantifier.

getHandle method

Purpose

Retrieve the handle of the quantifier. The handle is an integer value that is used by the federated server's query planning component.

Syntax

```
public int getHandle()
```

Parameters

None.

Return value

The quantifier handle.

Throws

None.

getNickname method

Purpose

Retrieve the nickname that is referenced by this quantifier.

Syntax

```
public UnfencedGenericNickname getNickname()
```

Parameters

None.

Return value

The nickname that is referenced by the quantifier.

Throws

None.

Related reference:

- "Reply class (Java)" on page 131
- "Request class (Java)" on page 130
- "UnfencedGenericNickname class (Java)" on page 99

PredicateList class (Java)

This topic describes the PredicateList class and provides details for the methods.

This class does not contain constructors.

Overview

The PredicateList class describes two lists of predicates to estimate the selectivity factor. Instances of this class are used as input to UnfencedGenericServer.getSelectivity, the selectivity estimation function. The two predicate lists are:

- a list of predicates that selectivity is solicited for (P)
- a list of previously applied predicates (AP)

The result selectivity is conditional selectivity: selectivity (P/AP). If you need unconditional selectivity, the AP can be null. If the implementation cannot support this framework, the implementation can be set to ignore the AP. The lists of predicates are similar to the list of predicates in the Request class and are manipulated with similar methods.

The PredicateList class is one of the planning classes for the Java API.

Package

com.ibm.db2.wrapper

Methods

The following table describe the methods of the PredicateList class. The methods are described in more detail after the table.

Table 177. Methods for the PredicateList class

Method	Description
getAppliedPredicate	Retrieve a RequestExp object that describes the applied predicate at the specified position.
getNumberOfAppliedPredicates	Retrieve the number of applied predicates.
getNumberOfPredicates	Retrieve the number of predicates in the list.
getPredicate	Retrieve a RequestExp object that describes the predicate at the specified position.

getAppliedPredicate method

Purpose

Retrieve a RequestExp object that describes the applied predicate at the specified position.

Syntax

```
public RequestExp getAppliedPredicate(int position)
```

Parameters

Table 178. Parameters for the getAppliedPredicate method

Name	Description
position	The predicate position. The first predicate is at position 1.

Return value
The applied predicate expression.

Throws
None.

getNumberOfAppliedPredicates method

Purpose
Retrieve the number of applied predicates.

Syntax

```
public int getNumberOfAppliedPredicates()
```

Parameters
None.

Return value
The number of applied predicates.

Throws
None.

getNumberOfPredicates method

Purpose
Retrieve the number of predicates in the list.

Syntax

```
public int getNumberOfPredicates()
```

Parameters
None.

Return value
The number of predicates.

Throws
None.

getPredicate method

Purpose
Retrieve a RequestExp object that describes the predicate at the specified position.

Syntax

```
public RequestExp getPredicate(int position)
```

Parameters

Table 179. Parameters for the getPredicate method

Name	Description
position	The predicate position. The first predicate is at possition 1.

Return value
The predicate expression.

Throws
None.

Related tasks:

- "Predicate list class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "RequestExp class (Java)" on page 139
- "WrapperException class (Java)" on page 190

Data classes for the Java API

The following table describes each data class for the Java API.

Table 180. Data classes

Class name	Description
Data	The class that manages cell values, constants, and parameter values; and provides methods to convert the values from internal formats into Java standard types and objects.
RuntimeData	The class that represents each column value that is transferred between the federated server and a wrapper.
RuntimeDataList	The class that encapsulates a list of RuntimeData objects that represent a list of input parameters or a row in a result set.
RuntimeDataDesc	The class that is created by the federated server to describe each column value that is transferred between the federated server and a wrapper.
RuntimeDataDescList	The class that encapsulates a list of RuntimeDataDesc objects that describe a list of parameter values or a row in a result set.

Related reference:

- "RuntimeDataDesc class (Java)" on page 181
- "RuntimeDataDescList class (Java)" on page 188
- "RuntimeDataList class (Java)" on page 179
- "RuntimeData class (Java)" on page 163

Data class (Java)

This topic describes the Data class and provides details for the methods.

This class does not contain any constructors.

Overview

The Data class is the base class that manages cell values, constants, and parameter values. This class provides methods to convert the values from internal formats into standard Java types and objects.

The RequestConstant class and the RuntimeData class are subclasses of the Data class.

The Data class is a data class for the Java API.

Package

com.ibm.db2.wrapper

Constants

The following table describes the valid constants for the options that you can use with the Data class.

Table 181. Constants for the Data class

Constant	Definition	Description
BLOB	public static final short BLOB	Constant that indicates a BLOB data type.
CHAR	public static final short CHAR	Constant that indicates a CHAR data type.
CLOB	public static final short CLOB	Constant that indicates a CLOB data type.
DATE	public static final short DATE	Constant that indicates a DATE data type.
DECIMAL	public static final short DECIMAL	Constant that indicates a DECIMAL or NUMERIC data type.
DOUBLE	public static final short DOUBLE	Constant that indicates a DOUBLE data type.
FLOAT	public static final short FLOAT	Constant that indicates a FLOAT data type.
INT	public static final short INT	Constant that indicates an INTEGER (INT) data type.
LONG	public static final short LONG	Constant that indicates a LONG data type.
NONE	public static final short NONE	Constant that indicates an unknown data type.
SHORT	public static final short SHORT	Constant that indicates a SHORT data type.
TIME	public static final short TIME	Constant that indicates a TIME data type.
TIMESTAMP	public static final short TIMESTAMP	Constant that indicates a TIMESTAMP data type.
VARCHAR	public static final short VARCHAR	Constant that indicates a VARCHAR data type.

Methods

The following table describes the methods of the Data class. The methods are described in more detail after the table.

Table 182. Methods for the Data class

Method	Description
getBigDecimal	Retrieve the data as a BigDecimal instance.
getByte	Retrieve the data as a byte value.
getData	Retrieve the data as a byte array.
getDataType	Retrieve the data type.
getDate	Retrieve the data as a Date instance.
getDouble	Retrieve the data as a DOUBLE data type value.
getFloat	Retrieve the data as a FLOAT data type value.
getForBitData	Retrieve the FOR BIT DATA flag.
getInt	Retrieve the data as an integer (INT) data type value.
getLong	Retrieve the data as a LONG data type value.
getObject	Retrieve the data as an Object instance.
getShort	Retrieve the data as a SHORT data type value.
getString	Retrieve the data as a string.
getTime	Retrieve the data as a Time instance.
getTimestamp	Retrieve the data as a Timestamp instance.

getBigDecimal method

Purpose

Retrieve the data as a BigDecimal instance.

Syntax

```
public java.math.BigDecimal getBigDecimal()
                                    throws WrapperException
```

Parameters

None.

Return value

The data.

Throws

A WrapperException object if the data type is not compatible.

getByte method

Purpose

Retrieve the data as a byte value.

Syntax

```
public byte getByte()
             throws WrapperException
```

Parameters
None.

Return value
The data.

Throws
A WrapperException object if the data type is not compatible.

getData method

Purpose
Retrieve the data as a byte array. The value is in an internal format.

Syntax

```
protected abstract byte[] getData()
```

Parameters
None.

Return value
The data.

Throws
None.

getDataType method

Purpose
Retrieve the data type.

Syntax

```
public abstract short getDataType()
```

Parameters
None.

Return value
The data type.

Throws
None.

getDate method

Purpose
Retrieve the data as a Date instance.

Syntax

```
public java.sql.Date getDate()
                        throws WrapperException
```

Parameters
None.

Return value
The data.

Throws
A WrapperException object if the data type is not compatible.

getDouble method

Purpose

Retrieve the data as a DOUBLE data type value.

Syntax

```
public double getDouble()
                  throws WrapperException
```

Parameters

None.

Return value

The data.

Throws

A WrapperException object if the data type is not compatible.

getFloat method

Purpose

Retrieve the data as a FLOAT data type value.

Syntax

```
public float getFloat()
                throws WrapperException
```

Parameters

None.

Return value

The data.

Throws

A WrapperException object if the data type is not compatible.

getForBitData method

Purpose

Retrieve the FOR BIT DATA flag. Use this method only for CHAR or LONG VARCHAR data types to indicate whether the data is stored in a binary format.

Syntax

```
public abstract boolean getForBitData()
```

Parameters

None.

Return value

The FOR BIT DATA flag value.

Throws

None.

getInt method

Purpose

Retrieve the data as an integer (INT) data type value.

Syntax

```
public int getInt()
           throws WrapperException
```

Parameters

None.

Return value

The data.

Throws

A WrapperException object if the data type is not compatible.

getLong method

Purpose

Retrieve the data as a LONG data type value.

Syntax

```
public long getLong()
            throws WrapperException
```

Parameters

None.

Return value

The data.

Throws

A WrapperException object if the data type is not compatible.

getObject method

Purpose

Retrieve the data as an Object instance.

Syntax

```
public java.lang.Object getObject()
                           throws WrapperException
```

Parameters

None.

Return value

The data.

Throws

A WrapperException object if the data type is not compatible.

getShort method

Purpose

Retrieve the data as a SHORT data type value.

Syntax

```
public short getShort()
                throws WrapperException
```

Parameters
None.

Return value
The data.

Throws
A WrapperException object if the data type is not compatible.

getString method

Purpose
Retrieve the data as a string.

Syntax

```
public java.lang.String getString()
                              throws WrapperException
```

Parameters
None.

Return value
The data.

Throws
A WrapperException object if the data type is not compatible.

getTime method

Purpose
Retrieve the data as a Time instance.

Syntax

```
public java.sql.Time getTime()
                       throws WrapperException
```

Parameters
None.

Return value
The data.

Throws
A WrapperException object if the data type is not compatible.

getTimestamp method

Purpose
Retrieve the data as a Timestamp instance.

Syntax

```
public java.sql.Timestamp getTimestamp()
                                  throws WrapperException
```

Parameters
None.

Return value
The data.

Throws
A WrapperException object if the data type is not compatible.

Related reference:
- "RequestConstant class (Java)" on page 147
- "RuntimeData class (Java)" on page 163
- "WrapperException class (Java)" on page 190

RuntimeData class (Java)

This topic describes the RuntimeData class and provides details for the methods.

This class does not contain constructors.

Overview

The RuntimeData class represents each column value that is transferred between the federated server and a wrapper. A column value can be part of a result row that is transferred from the wrapper to the federated server, or it can be a value to be bound to a runtime parameter in a query that is submitted to the wrapper by the federated server.

The public RuntimeData class extends the Data class.

The RuntimeData class is one of the data classes for the Java API.

Package
com.ibm.db2.wrapper

Constants and class fields

The following method constants and class fields are inherited from the Data class and can be used with the RuntimeData class:
- BLOB
- CHAR
- CLOB
- DATE
- DECIMAL
- DOUBLE
- FLOAT
- INT
- LONG
- NONE
- SHORT
- TIME
- TIMESTAMP
- VARCHAR

Methods

The following table describes the methods of the RuntimeData class. The methods are described in more detail after the table.

Table 183. Methods for the RuntimeData class

Method	Description
checkFriendlyDivBy	Determine if the reason for a null indication is a divide by zero error.
checkFriendlyException	Determine if the reason for a null indication is a numeric exception.
clearNullIndicator	Clear the null indicator for the data value.
getActualLength	Retrieve the actual length for the data value.
getCodepage	Retrieve the code page for character data type values.
getData	Retrieve the data value in an internal format.
getDataIndex	Retrieve the column number for the data value.
getDataType	Retrieve the data type.
getForBitData	Retrieve the FOR BIT DATA flag, which indicates binary data.
getInvariant	Retrieve the invariant value.
getMaximumLength	Retrieve the maximum length of the data.
getName	Retrieve the name for the data.
getNullIndicator	Retrieve the null indicator for the data value.
getPrecision	Retrieve the precision for numeric data type values.
getRemoteLength	Retrieve the remote length of the data value.
getRemotePrecision	Retrieve the remote precision for numeric data type values.
getRemoteScale	Retrieve the remote scale for numeric data type values.
getRemoteType	Retrieve the remote type of the data value.
getScale	Retrieve the scale for numeric data type values.
isDataNull	Indicate whether the data value is null or is not null.
isDataNullable	Indicate whether the data value is nullable or is not nullable.
isSemanticNull	Indicate whether the data value is semantic null or is not semantic null.
setActualLength	Set the actual length for the data value.
setBigDecimal	Set the data value as a BigDecimal instance.
setBinary	Set the data value as a binary value.
setByte	Set the data value as a byte.
setDataNull	Mark the data value as null.
setDate	Set the data value as a Date instance.

Table 183. Methods for the RuntimeData class (continued)

Method	Description
setDouble	Set the data as a double data type value.
setFloat	Set the data as a float data type value.
setFriendlyDivBy0	Indicate that a value is null because a divide by zero error occurred.
setFriendlyException	Indicate that a value is null because of a numeric exception.
setInt	Set the data as an int data type value.
setLong	Set the data as a long data type value.
setNullIndicator	Set the null indicator for the data value.
setObject	Set the data value as an Object instance.
setPrecision	Set the precision for numeric data type values.
setScale	Set the scale for numeric data type values.
setShort	Set the data as a short data type value.
setString	Set the data as a string value.
setTime	Set the data as a Time instance value.
setTimestamp	Set the data as a Timestamp instance value.

checkFriendlyDivBy method

Purpose

Determine if the reason for a null indication is a divide by zero error.

Syntax

```
public boolean checkFriendlyDivBy0()
```

Parameters

None.

Return value

A value of true if a divide by zero error occurred. Otherwise, the value is false.

Throws

None.

checkFriendlyException method

Purpose

Determine if the reason for a null indication is a numeric exception.

Syntax

```
public boolean checkFriendlyException()
```

Parameters

None.

Return value

A value of true if the reason for a null indication is a numeric exception. Otherwise, the value is false.

Throws
None.

clearNullIndicator method

Purpose
Clear the null indicator for the data value.

Syntax

```
public void clearNullIndicator()
                          throws WrapperException
```

Parameters
None.

Return value
None.

Throws
A WrapperException object if the operation fails.

getActualLength method

Purpose
Retrieve the actual length for the data value.

Syntax

```
public int getActualLength()
```

Parameters
None.

Return value
None.

Throws
A WrapperException object if the operation fails.

getData method

Purpose
Retrieve the data value in the internal format of the federated server.

Syntax

```
protected byte[] getData()
```

Overrides
The getData method in the Data class.

Parameters
None.

Return value
The data in the internal format of the federated server.

Throws
None.

getDataIndex method

Purpose

Retrieve the column number for the data value.

Syntax

```
public int getDataIndex()
```

Parameters

None.

Return value

The column number.

Throws

None.

getDataType method

Purpose

Retrieve the data type.

Syntax

```
public short getDataType()
```

Overrides

The getDataType method in the Data class.

Parameters

None.

Return value

The data type.

Throws

None.

getForBitData method

Purpose

Retrieve the FOR BIT DATA flag, which indicates binary data.

Syntax

```
public short getDataType()
```

Overrides

The getForBitData method in the Data class.

Parameters

None.

Return value

The FOR BIT DATA data flag.

Throws

None.

getInvariant method

Purpose

Retrieve the invariant value. The value of an invariant parameter does not

change unless the wrapper is notified by a change to the action parameter of the RemoteQuery.reopen(short) method.

Syntax

```
public byte getInvariant()
```

Parameters

None.

Return value

The invariant value.

Throws

None.

getMaximumLength method

Purpose

Retrieve the maximum length of the data.

Syntax

```
public int getMaximumLength()
```

Parameters

None.

Return value

The maximum length.

Throws

None.

getName method

Purpose

Retrieve the name for the data.

Syntax

```
public java.lang.String getName()
```

Parameters

None.

Return value

The name.

Throws

None.

getNullIndicator method

Purpose

Retrieve the null indicator for the data value.

Syntax

```
public short getNullIndicator()
```

Parameters

None.

Return value
The null indicator.

Throws
None.

getPrecision method

Purpose
Retrieve the precision for numeric data type values.

Syntax

```
public byte getPrecision()
```

Parameters
None.

Return value
The precision.

Throws
None.

getRemoteLength method

Purpose
Retrieve the remote length of the data value.

Syntax

```
public int getRemoteLength()
```

Parameters
None.

Return value
The remote length.

Throws
None.

getRemotePrecision method

Purpose
Retrieve the remote precision of numeric data type values.

Syntax

```
public byte getRemotePrecision()
```

Parameters
None.

Return value
The remote precision.

Throws
None.

getRemoteScale method

Purpose

Retrieve the remote scale of numeric data type values.

Syntax

```
public byte getRemoteScale()
```

Parameters

None.

Return value

The remote scale.

Throws

None.

getRemoteType method

Purpose

Retrieve the remote type of the data value.

Syntax

```
public short getRemoteType()
```

Parameters

None.

Return value

The remote type.

Throws

None.

getScale method

Purpose

Retrieve the scale for numeric data type values.

Syntax

```
public byte getScale()
```

Parameters

None.

Return value

The scale.

Throws

None.

isDataNull method

Purpose

Indicate whether the data value is null or is not null.

Syntax

```
public boolean isDataNull()
```

Parameters

None.

Retrieve value
A value of true if the data is null. Otherwise, the value is false.

Throws
None.

isDataNullable method

Purpose
Indicate whether the data value is nullable or is not nullable.

Syntax

```
public boolean isDataNullable()
```

Parameters
None.

Return value
A value of true if the data is nullable. Otherwise, the value is false.

Throws
None.

isSemanticNull method

Purpose
Indicate whether the data value is semantic null or is not semantic null.

Syntax

```
public boolean isSemanticNull()
```

Parameters
None.

Return value
A value of true if the data is semantic null. Otherwise, the value is false.

Throws
None.

setActualLength method

Purpose
Set the actual length for the data value.

Syntax

```
public void setActualLength(int length)
```

Parameters

Table 184. Parameters for the setActualLength method

Name	Description
length	Actual length.

Return value
None.

Throws
None.

setBigDecimal method

Purpose

Set the data value as a BigDecimal instance.

Syntax

```
public void setBigDecimal(java.math.BigDecimal value)
                    throws WrapperException
```

Parameters

Table 185. Parameters for the setBigDecimal method

Name	Description
value	The data value.

Return value

None.

Throws

A WrapperException object if the data is not compatible with the data type.

setBinary method

Purpose

Set the data value as a binary value.

Syntax

```
public void setBinary(byte[] value)
                throws WrapperException
```

Parameters

Table 186. Parameters for the setBinary method

Name	Description
value	The data value.

Return value

None.

Throws

A WrapperException object if the data is not compatible with the data type.

setByte method

Purpose

Set the data value as a byte.

Syntax

```
public void setByte(byte value)
              throws WrapperException
```

Parameters

Table 187. Parameters for the setByte method

Name	Description
value	The data value.

Return value
None.

Throws
A WrapperException object if the data is not compatible with the data type.

setDataNull method

Purpose
Mark the data value as null.

Syntax

```
public void setDataNull()
                throws WrapperException
```

Parameters
None.

Return value
None.

Throws
A WrapperException object if the data cannot be null.

setDate method

Purpose
Set the data value as a Date instance.

Syntax

```
public void setDate(java.sql.Date value)
             throws WrapperException
```

Parameters

Table 188. Parameters for the setDate method

Name	Description
value	The data value.

Return value
None.

Throws
A WrapperException object if the data is not compatible with the data type.

setDouble method

Purpose
Set the data as a double data type value.

Syntax

```
public void setDouble(double value)
                throws WrapperException
```

Parameters

Table 189. Parameters for the setDouble method

Name	Description
value	The data value.

Return value
None.

Throws
A WrapperException object if the data is not compatible with the data type.

setFloat method

Purpose
Set the data as a float data type value.

Syntax

```
public void setFloat(float value)
              throws WrapperException
```

Parameters

Table 190. Parameters for the setFloat method

Name	Description
value	The data value.

Return value
None.

Throws
A WrapperException object if the data is not compatible with the data type.

setFriendlyDivBy0 method

Purpose
Indicate that a value is null because a divide by zero error occurred.

Usage Verify that your wrapper calls the setDataNull() method before the wrapper calls the setFriendlyDivBy0 method.

Syntax

```
public void setFriendlyDivBy0()
                      throws WrapperException
```

Parameters
None.

Return value
None.

Throws
A WrapperException object if the operation fails.

setFriendlyException method

Purpose

Indicate that a value is null because of a numeric exception.

Usage Verify that your wrapper calls the setDataNull() method before the wrapper calls the setFriendlyException method.

Syntax

```
public void setFriendlyException()
                              throws WrapperException
```

Parameters

None.

Return value

None.

Throws

A WrapperException object if the operation fails.

setInt method

Purpose

Set the data as an int data type value.

Syntax

```
public void setInt(int value)
            throws WrapperException
```

Parameters

Table 191. Parameters for the setInt method

Name	Description
value	The data value.

Return value

None.

Throws

A WrapperException object if the data is not compatible with the data type.

setLong method

Purpose

Set the data as a long data type value.

Syntax

```
public void setLong(long value)
             throws WrapperException
```

Parameters

Table 192. Parameters for the setLong method

Name	Description
value	Data value to be set.

Return value

None.

Throws

A WrapperException object if the data is not compatible with the data type.

setNullIndicator method

Purpose

Set the null indicator for the data value.

Syntax

```
public void setNullIndicator(short indicator)
```

Parameters

Table 193. Parameters for the setNullIndicator method

Name	Description
indicator	Null indicator.

Return value

None.

Throws

None.

setObject method

Purpose

Set the data value as an Object instance.

Syntax

```
public void setObject(java.lang.Object value)
                throws WrapperException
```

Parameters

Table 194. Parameters for the setObject method

Name	Description
value	Data value to be set.

Return value

None.

Throws

A WrapperException object if the data is not compatible with the data type.

setPrecision method

Purpose

Set the precision for numeric data type values.

Syntax

```
public void setPrecision(byte precision)
```

Parameters

Table 195. Parameters for the setPrecision method

Name	Description
precision	Precision to be set.

Return value
None.

Throws
None.

setScale method

Purpose
Set the scale for numeric data type values.

Syntax

```
public void setScale(byte scale)
```

Parameters

Table 196. Parameters for the setScale method

Name	Description
scale	Scale to be set.

Return value
None.

Throws
None.

setShort method

Purpose
Set the data as a short data type value.

Syntax

```
public void setShort(short value)
                throws WrapperException
```

Parameters

Table 197. Parameters for the setShort method

Name	Description
value	Data value to be set.

Return value
None.

Throws
A WrapperException object if the data is not compatible with the data type.

setString method

Purpose

Set the data value as a string.

Syntax

```
public void setString(java.lang.String value)
                throws WrapperException
```

Parameters

Table 198. Parameters for the setString method

Name	Description
value	Data value to be set.

Return value

None.

Throws

A WrapperException object if the data is not compatible with the data type.

setTime method

Purpose

Set the data value as a Time instance.

Syntax

```
public void setTime(java.sql.Time value)
             throws WrapperException
```

Parameters

Table 199. Parameters for the setTime method

Name	Description
value	Data value to be set.

Return value

None.

Throws

A WrapperException object if the data is not compatible with the data type.

setTimestamp method

Purpose

Set the data value as a Timestamp instance.

Syntax

```
public void setTimestamp(java.sql.Timestamp value)
                  throws WrapperException
```

Parameters

Table 200. Parameters for the setTimestamp method

Name	Description
value	Data value to be set.

Return value
None.

Throws
A WrapperException object if the data is not compatible with the data type.

Inherited methods

The following methods are inherited from the Data class and can be used with the RuntimeData class:

- getBigDecimal
- getByte
- getDate
- getDouble
- getFloat
- getInt
- getLong
- getObject
- getShort
- getString
- getTime
- getTimestamp

Related tasks:

- "Runtime data classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "Data class (Java)" on page 156
- "RemoteQuery class (Java)" on page 115
- "WrapperException class (Java)" on page 190

RuntimeDataList class (Java)

This topic describes the RuntimeDataList class and provides details for the methods.

This class does not contain constructors.

Overview

The RuntimeDataList class encapsulates a list of RuntimeData objects that represent either a row in a result set or a list of input parameters.

The RuntimeDataList class is one of the data classes for the Java API.

Package
com.ibm.db2.wrapper

Methods

The following table describes the methods of the RuntimeDataList class. These methods are described in more detail after the table.

Table 201. Methods for the RuntimeDataList class

Method	Description
getNumberOfValues	Retrieve the number of values in the list.
getValue	Retrieve the value at the specified position in the list.

getNumberOfValues method

Purpose

Retrieve the number of values in the list.

Syntax

```
public int getNumberOfValues()
```

Parameters

None.

Return value

The number of values.

Throws

None.

getValue method

Purpose

Retrieve the value at the specified position in the list.

Syntax

```
public RuntimeData getValue(int position)
```

Parameters

Table 202. Parameters for the getValue method

Name	Description
position	Position of the value to be returned. The first value is at position zero in the list.

Return value

The data value object.

Throws

None.

Related tasks:

- "Runtime data classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "RuntimeDataDesc class (Java)" on page 181

- "RuntimeData class (Java)" on page 163

RuntimeDataDesc class (Java)

This topic describes the RuntimeDataDesc class and provides details for the constructors and the methods.

Overview

The federated server creates the RuntimeDataDesc class to describe each column value that is transferred between the federated server and a wrapper. A column value can be either one of the following items:

- A portion of a result row that is transferred from the wrapper to the federated server.
- A value to be bound to a runtime parameter in a query that is submitted to the wrapper by the federated server.

The RuntimeDataDesc class is one of the data classes for the Java API.

Usage For both RemoteQuery and RemotePassthru objects, the federated server creates a RuntimeDataDesc instance for each RuntimeData object in the input data list that represents the parameter values. The federated server supplies the column descriptions. For RemoteQuery objects, the federated server also creates a RuntimeDataDesc instance for each RuntimeData object in the output data list that represents values in result rows. The federated server supplies these column descriptions as well. The wrapper writer creates instances of the RuntimeDataDesc class to describe the result set of a pass-through session in the RemotePassthru.describe method.

Package

com.ibm.db2.wrapper

Constructors and methods

The following table describes the constructors and methods of the RuntimeDataDesc class. These constructors and methods are described in more detail after the tables.

Table 203. Constructors for the RuntimeDataDesc class

Constructor	Description
RuntimeDataDesc	Construct a new RuntimeDataDesc object with the specified attributes (type ID, maximum length, code page, and null indicator) and describe a column of a result set.
RuntimeDataDesc	Construct a new RuntimeDataDesc object with the specified attributes (type ID, maximum length, code page, null indicator, and precision) and describe a column of a result set.
RuntimeDataDesc	Construct a new RuntimeDataDesc object with the specified attributes (type ID, maximum length, code page, null indicator, precision, and scale) and describe a column of a result set.

Table 203. Constructors for the RuntimeDataDesc class (continued)

Constructor	Description
RuntimeDataDesc	Construct a new RuntimeDataDesc object with the specified attributes type ID, maximum length, code page, null indicator, precision, scale, and the name of the data column) and describe a column of a result set.
RuntimeDataDesc	Construct a new RuntimeDataDesc object with the specified attributes (type ID, maximum length, code page, null indicator, precision, scale, the name of the data column, and the type of remote data) and describe a column of a result set.

Table 204. Methods for the RuntimeDataDesc class

Method	Description
getCodepage	Retrieve the code page for character data type values.
getDataType	Retrieve the data type.
getForBitData	Retrieve the FOR BIT DATA flag, which indicates binary data.
getMaximumLength	Retrieve the maximum length of the data.
getName	Retrieve the name for the data.
getNullIndicator	Retrieve the null indicator for the data value.
getPrecision	Retrieve the precision for numeric data type values.
getRemoteType	Retrieve the remote type of the data.
getScale	Retrieve the scale for numeric data type values.

RuntimeDataDesc constructor

Purpose

Construct a new RuntimeDataDesc object with the specified attributes and describe a column of a result set.

Syntax

```
public RuntimeDataDesc(short type,
                       int maxLength,
                       short codepage,
                       short nullIndicator)
                 throws WrapperException
```

Parameters

Table 205. Parameters for the RuntimeDataDesc constructor

Name	Description
type	Type ID for the data.
maxLength	Maximum length of the data.
codepage	Code page that represents the data. This parameter is valid for character data types only.

Table 205. Parameters for the RuntimeDataDesc constructor (continued)

Name	Description
nullIndicator	Indicator of a null value.

Throws

A WrapperException object if the method fails.

RuntimeDataDesc constructor

Purpose

Construct a new RuntimeDataDesc object with the specified attributes and describe a column of a result set.

Syntax

```
public RuntimeDataDesc(short type,
                       int maxLength,
                       short codepage,
                       short nullIndicator,
                       byte precision)
                 throws WrapperException
```

Parameters

Table 206. Parameters for the RuntimeDataDesc constructor

Name	Description
type	Type ID for the data.
maxLength	Maximum length of the data.
codepage	Code page that represents the data. This parameter is valid for character data types only.
nullIndicator	Indicator of a null value.
precision	Precision of the data. This parameter is valid for numeric and decimal data types only.

Throws

A WrapperException object if the method fails.

RuntimeDataDesc constructor

Purpose

Construct a new RuntimeDataDesc object with the specified attributes and describe a column of a result set.

Syntax

```
public RuntimeDataDesc(short type,
                       int maxLength,
                       short codepage,
                       short nullIndicator,
                       byte precision,
                       byte scale)
                 throws WrapperException
```

Parameters

Table 207. Parameters for the RuntimeDataDesc constructor

Name	Description
type	Type ID for the data.
maxLength	Maximum length of the data.
codepage	Code page that represents the data. This parameter is valid for character data types only.
nullIndicator	Indicator of a null value.
precision	Precision of the data. This parameter is valid for numeric and decimal data types only.
scale	Scale of the data. This parameter is valid for numeric and decimal data types only.

Throws

A WrapperException object if the method fails.

RuntimeDataDesc constructor

Purpose

Construct a new RuntimeDataDesc object with the specified attributes and describe a column of a result set.

Syntax

```
public RuntimeDataDesc(short type,
                       int maxLength,
                       short codepage,
                       short nullIndicator,
                       byte precision,
                       byte scale,
                       java.lang.String name)
                 throws WrapperException
```

Parameters

Table 208. Parameters for the RuntimeDataDesc constructor

Name	Description
type	Type ID for the data.
maxLength	Maximum length of the data.
codepage	Code page that represents the data. This parameter is valid for character data types only.
nullIndicator	Indicator of a null value.
precision	Precision of the data. This parameter is valid for numeric and decimal data types only.
scale	Scale of the data. This parameter is valid for numeric and decimal data types only.
name	Name of the data column.

Throws

A WrapperException object if the method fails.

RuntimeDataDesc constructor

Purpose

Construct a new RuntimeDataDesc object with the specified attributes and describe a column of a result set.

Syntax

```
public RuntimeDataDesc(short type,
                       int maxLength,
                       short codepage,
                       short nullIndicator,
                       byte precision,
                       byte scale,
                       java.lang.String name,
                       short remoteType)
                 throws WrapperException
```

Parameters

Table 209. Parameters for the RuntimeDataDesc constructor

Name	Description
type	Type ID for the data.
maxLength	Maximum length of the data.
codepage	Code page that represents the data. This parameter is valid for character data types only.
nullIndicator	Indicator of a null value.
precision	Precision of the data. This parameter is valid for numeric and decimal data types only.
scale	Scale of the data. This parameter is valid for numeric and decimal data types only.
name	Name of the data column.
remoteType	Type of the data at the remote data source.

Throws

A WrapperException object if the method fails.

getCodepage method

Purpose

Retrieve the code page for character data type values.

Syntax

```
public short getCodepage()
```

Parameters

None.

Return value

The code page.

Throws

None.

getDataType method

Purpose

Retrieve the data type.

Syntax

```
public short getDataType()
```

Parameters

None.

Return value

The data type.

Throws

None.

getForBitData method

Purpose

Retrieve the FOR BIT DATA flag, which indicates binary data.

Syntax

```
public boolean getForBitData()
```

Overrides

The getForBitData method in the Data class.

Parameters

None.

Return value

The FOR BIT DATA data flag.

Throws

None.

getMaximumLength method

Purpose

Retrieve the maximum length of the data.

Syntax

```
public int getMaximumLength()
```

Parameters

None.

Return value

The maximum length.

Throws

None.

getName method

Purpose

Retrieve the name for the data.

Syntax

```
public java.lang.String getName()
                            throws WrapperException
```

Parameters
None.

Return value
The name.

Throws
A WrapperException object if the method fails.

getNullIndicator method

Purpose
Retrieve the null indicator for the data value.

Syntax

```
public short getNullIndicator()
```

Parameters
None.

Return value
The null indicator.

Throws
None.

getPrecision method

Purpose
Retrieve the precision for numeric data type values.

Syntax

```
public byte getPrecision()
```

Parameters
None.

Return value
The precision.

Throws
None.

getRemoteType method

Purpose
Retrieve the remote type of the data value.

Syntax

```
public short getRemoteType()
```

Parameters
None.

Return value
The remote type.

Throws
None.

getScale method

Purpose
Retrieve the scale for numeric data type values.

Syntax

```
public byte getScale()
```

Parameters
None.

Return value
The scale.

Throws
None.

Related tasks:

- "Runtime data description classes" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "RemotePassthru class (Java)" on page 111
- "RemoteQuery class (Java)" on page 115
- "RuntimeData class (Java)" on page 163
- "WrapperException class (Java)" on page 190

RuntimeDataDescList class (Java)

This topic describes the RuntimeDataDescList class and provides details for the methods.

This class does not contain constructors.

Overview

The RuntimeDataDescList class encapsulates a list of RuntimeDataDesc objects that describe a row in a result set or a list of input parameters.

The RuntimeDataDesc class is one of the data classes for the Java API.

Package
com.ibm.db2.wrapper

Methods

The following table describes the methods of the RuntimeDataDescList class. These methods are described in more detail after the table.

Table 210. methods for the RuntimeDataDescList class

Method	Description
getNumberOfValues	Retrieve the number of data descriptors in the list.
getValue	Retrieve the data descriptor at the specified position in the list.
setNumberOfValues	Set the number of data descriptors in the list.
setValue	Set the data descriptor at the specified position in the list.

getNumberOfValues method

Purpose

Retrieve the number of data descriptors in the list.

Syntax

```
public int getNumberOfValues()
```

Parameters

None.

Return value

The number of data descriptors .

Throws

None.

getValue method

Purpose

Retrieve the data descriptor at the specified position in the list.

Syntax

```
public RuntimeDataDesc getValue(int position)
```

Parameters

Table 211. Parameters for the getValue method

Name	Description
position	Position of the data descriptor to be returned. The first data descriptor is at position zero in the list.

Return value

The data descriptor.

Throws

None.

setNumberOfValues method

Purpose

Set the number of data descriptors in the list.

Syntax

```
public void setNumberOfValues(int newNumber)
                        throws WrapperException
```

Parameters

Table 212. Parameters for the setNumberOfValues method

Name	Description
newNumber	Number of data descriptors.

Return value
None.

Throws
A WrapperException object if the method fails.

setValue method

Purpose
Set the data descriptor at the specified position in the list.

Syntax

```
public void setValue(RuntimeDataDesc runtimeDataDesc,
                     int position)
              throws WrapperException
```

Parameters

Table 213. Parameters for the setValue method

Name	Description
runtimeDataDesc	Data descriptor that is saved in the list.
position	Position where the data descriptor is placed. The first data descriptor is at position zero in the list.

Return value
None.

Throws
A WrapperException object if the operation fails.

Related reference:
- "RuntimeDataDesc class (Java)" on page 181
- "WrapperException class (Java)" on page 190

WrapperException class (Java)

This topic describes the WrapperException class and provides details for the constructors and the methods.

Overview

The WrapperException class is the exception class that is used by the Java API to report exceptions.

The public WrapperException class extends the java.lang.Exception class.

Usage This class is instantiated by the wrapper to report an exception that contains either a string message, or an SQL error code, caller function name, and tokens that map to a DB2 Information Integrator error message.

Package
com.ibm.db2.wrapper

Constructors and methods

The following tables describe the constructors and the methods of the WrapperException class. The constructors and methods are described in more detail after the tables.

Table 214. Constructors for the WrapperException class

Constructor	Description
WrapperException	Construct an exception object that reports a DB2 Information Integrator error by its code, caller function name, and tokens.
WrapperException	Construct an exception object with a specified message.

Table 215. Methods for the WrapperException class

Method	Description
getAndResetErrorCode	Retrieve and reset the SQL error code to report a DB2 Information Integrator error.
getAndResetFunctionName	Retrieve and reset the caller function name that reports a DB2 Information Integrator error.
getAndResetTokens	Retrieve and reset the substitution tokens that are used to report a DB2 Information Integrator error.
getErrorCode	Retrieve the SQL error code to report a DB2 Information Integrator error.
getFunctionName	Retrieve the caller function name that reports a DB2 Information Integrator error.
getMessage	Retrieve the error message string of this exception object.
getStackTrace	Save the stack trace of the exception into a string.
getTokens	Retrieve the substitution tokens for reporting a DB2 Information Integrator error.
setErrorCode	Set the SQL error code for reporting a DB2 Information Integrator error.
setFunctionName	Set the caller function name for reporting a DB2 Information Integrator error.
setTokens	Set the substitution tokens for reporting a DB2 Information Integrator error.

WrapperException constructor

Purpose
Construct an exception object that reports a DB2 Information Integrator error by its SQL code, caller function name, and tokens. Each valid SQL

code identifies an error message. The error message might contain placeholders that are replaced with the given tokens before the message is reported to the user.

Syntax

```
public WrapperException(int errorCode,
                        java.lang.String functionName,
                        java.lang.String[] tokens)
```

Parameters

Table 216. Parameters for the WrapperException constructor

Name	Description
errorCode	Predefined SQL code of the error that is reported.
functionName	Name of the function that reports the error. The string value cannot be greater than five characters. The client program can access this string value through the SQLERRP field of the SQLCA. The string value is in uppercase letters with a prefix of SQL.
tokens	Substitution tokens for the message.

WrapperException constructor

Purpose

Construct the exception object with a specified message. If an error code is not specified in the exception that is thrown, an SQL0901 error is reported, and the specified exception message replaces the SQL0901 error message placeholder.

Syntax

```
public WrapperException(java.lang.String message)
```

Parameters

Table 217. Parameters for the WrapperException constructor

Name	Description
message	Message that describes the exception.

getAndResetErrorCode method

Purpose

Retrieve and reset the SQL error code to report a DB2 Information Integrator error. Each DB2 Information Integrator error is identified by an SQL error code.

Syntax

```
public final int getAndResetErrorCode()
```

Parameters

None.

Return value

The predefined SQL code of the error that is reported.

Throws

None.

getAndResetFunctionName method

Purpose

Retrieve and reset the caller function name that reports a DB2 Information Integrator error. The caller function name is reported with a DB2 error in the SQLERRP field of the SQLCA structure.

Syntax

```
public final java.lang.String getAndResetFunctionName()
```

Parameters

None.

Return value

The name of the function that reports the error. The string value cannot be greater than five characters. The client program can access this string value through the SQLERRP field of the SQLCA. The string value is in uppercase letters with a prefix of SQL.

Throws

None.

getAndResetTokens method

Purpose

Retrieve and reset the substitution tokens that are used to report a DB2 Information Integrator error. The substitution tokens replace the placeholders in the DB2 Information Integrator error message.

Syntax

```
public final java.lang.String[] getAndResetTokens()
```

Parameters

None.

Return value

The substitution tokens that are used in the error message when reporting the DB2 Information Integrator error.

Throws

None.

getErrorCode method

Purpose

Retrieve the SQL error code to report a DB2 Information Integrator error. Each DB2 Information Integrator error is identified by a SQL error code.

Syntax

```
public final int getErrorCode()
```

Parameters

None.

Return value

The predefined SQL code of the error that is reported.

Throws

None.

getFunctionName method

Purpose

Retrieve the caller function name that reports a DB2 Information Integrator error. The caller function name is reported with a DB2 Information Integrator error in the SQLERRP field of the SQLCA structure.

Syntax

```
public final java.lang.String getFunctionName()
```

Parameters

None.

Return value

The name of the function that reports the error. The string value cannot be greater than five characters. The client program can access this string value through the SQLERRP field of the SQLCA. The string value is in uppercase letters with a prefix of SQL.

Throws

None.

getMessage method

Purpose

Retrieve the error message string of this exception object.

Syntax

```
public java.lang.String getMessage()
```

Parameters

None.

Return value

If the WraperException object was created with an error message string, the error message string of this WrapperException object. If the WraperException object was created with an SQL error code, a string showing the SQL error code, the caller function name, and a set of tokens.

Throws

None.

getStackTrace method

Purpose

Save the stack trace of the exception into a string.

Syntax

```
public static java.lang.String getStackTrace(java.lang.Throwable throwable)
```

Parameters

Table 218. Parameters for the getStackTrace method

Name	Description
throwable	Throwable object that the stack trace is extracted from.

Return value

The string that contains the stack trace information.

Throws

None.

getTokens method

Purpose

Retrieve the substitution tokens that are used to report a DB2 Information Integrator error. The substitution tokens replace the placeholders in the DB2 Information Integrator error message.

Syntax

```
public final java.lang.String[] getTokens()
```

Parameters

None.

Return value

The substitution tokens that are used in the error message when a DB2 Information Integrator error is reported.

Throws

None.

setErrorCode method

Purpose

Set the SQL error code for reporting a DB2 Information Integrator error. Each DB2 Information Integrator error is identified by a SQL error code.

Syntax

```
public final void setErrorCode(int errorCode)
```

Parameters

Table 219. Parameters for the setErrorCode method

Name	Description
errorCode	Predefined SQL code of the error that is reported.

Return value

None.

Throws

None.

setFunctionName method

Purpose

Set the caller function name for reporting a DB2 Information Integrator error. The caller function name is reported with a DB2 Information Integrator error in the SQLERRP field of the SQLCA structure.

Syntax

```
public final void setFunctionName(java.lang.String functionName)
```

Parameters

Table 220. Parameters for the setFunctionName method

Name	Description
functionName	Name of the function that reports the error. The string value cannot be greater than five characters. The client program can access this string value through the SQLERRP field of the SQLCA. The string value is in uppercase letters with a prefix of SQL.

Return value

None.

Throws

None.

setTokens method

Purpose

Set the substitution tokens that are used to report a DB2 Information Integrator error. The substitution tokens replace the placeholders in the DB2 Information Integrator error message.

Syntax

```
public final void setTokens(java.lang.String[] tokens)
```

Parameters

Table 221. Parameters for the setTokens method

Name	Description
tokens	Substitution tokens that are used in the error message when a DB2 Information Integrator error is reported.

Return value

None.

Throws

None.

Related reference:

• "WrapperUtilities class (Java)" on page 197

WrapperUtilities class (Java)

This topic describes the WrapperUtilities class and provides details for the methods.

This class does not contain constructors.

Overview

The WrapperUtilities class is a container for several static utility functions.

The WrapperUtilities class is a utilities class for the Java API.

Usage Do not instantiate or subclass the WrapperUtilities class.

Package

com.ibm.db2.wrapper

Constants

The following table describes the valid constants for the options that you can use with the WrapperUtilities class.

Table 222. Constants for the WrapperUtilities class

Constant	Example	Description
EXT_WRAPPERS_ TRACE_ COMPONENT	public static final int EXT_WRAPPERS_TRACE_ COMPONENT	Trace component ID for wrappers not provided by IBM.

Methods

The following table describes the methods of the WrapperUtilities class. The methods are described in more detail after the table.

Table 223. Methods for the WrapperUtilities class

Method	Description
getAppCBStatus	Retrieve the status of the application control block.
getAuthid	Retrieve the authorization ID for the current database.
getCodepage	Retrieve the current database code page.
getDB2InstallPath	Retrieve the DB2 Universal Database installation path.
getDB2InstancePath	Retrieve the DB2 Universal Database instance path.
getDB2Release	Return the DB2 Universal Database release and the fix pack that this wrapper is currently running under.
getDoubleByteDBCodepage	Return the double-byte database code page.
getIsolationLevel	Return the isolation level for the current database.
getSingleByteDBCodepage	Returns the single-byte database code page.
reportWarning	Report a warning to the DB2 Information Integrator user.

Table 223. Methods for the WrapperUtilities class (continued)

Method	Description
traceEnabled	Verify whether DB2 Universal Database tracing is enabled.
traceError	Trace an error message.
traceException	Trace an exception and its call stack.
traceFunctionData	Trace function data using a single data trace parameter.
traceFunctionData	Trace function data using two data trace parameters.
traceFunctionData	Trace function data using three data trace parameters.
traceFunctionEntry	Trace a function entry.
traceFunctionReturnCode	Trace a function return code.

getAppCBStatus method

Purpose

Retrieve the status of the application control block.

Syntax

```
public static final int getAppCBStatus()
```

Parameters

None.

Return value

The application control block status.

Throws

None.

getAuthid method

Purpose

Retrieve the authorization ID for the current database.

Syntax

```
public static final java.lang.String getAuthid()
                                          throws WrapperException
```

Parameters

None.

Return value

The database authorization ID.

Throws

A WrapperException object if the operation fails.

getCodepage method

Purpose

Retrieve the current database code page.

Syntax

```
public static final int getCodepage()
```

Parameters
None.

Return value
The code page.

Throws
None.

getDB2InstallPath method

Purpose
Retrieve the DB2 Universal Database installation path.

Syntax

```
public static final java.lang.String getDB2InstallPath()
                                                    throws WrapperException
```

Parameters
None.

Return value
The DB2 information Integrator installation path.

Throws
A WrapperException object if the operation fails.

getDB2InstancePath method

Purpose
Retrieve the DB2 Universal Database instance path.

Syntax

```
public static final java.lang.String getDB2InstancePath()
                                                     throws WrapperException
```

Parameters
None.

Return value
The DB2 Information Integrator instance path.

Throws
A WrapperException object if the operation fails.

getDB2Release method

Purpose
Retrieve the DB2 Universal Database release and the fix pack that this wrapper is currently running under. This value will be updated at each DB2 Universal Database fix pack.

Syntax

```
public static final int getDB2Release()
```

Parameters
None.

Return value
The DB2 Information Integrator release.

Throws
None.

getDoubleByteDBCodepage method

Purpose
Retrieve the double byte database code page.

Syntax

```
public static final int getDoubleByteDBCodepage()
```

Parameters
None.

Return value
The double byte code page.

Throws
None.

getIsolationLevel method

Purpose
Retrieve the isolation level for the current database.

Syntax

```
public static final int getIsolationLevel()
```

Parameters
None.

Return value
The isolation level.

Throws
None.

getSingleByteDBCodepage method

Purpose
Retrieve the single byte database code page.

Syntax

```
public static final int getSingleByteDBCodepage()
```

Parameters
None.

Return value
The single byte code page.

Throws
None.

reportWarning method

Purpose

Report a warning to the DB2 Information Integrator user. A DB2 Information Integrator warning is composed of an SQL code, the caller function name, and a set of tokens. Each valid SQL code identifies a warning message. The warning message might contain placeholders, which are replaced with the specified tokens before the message is reported to the user.

Syntax

```
public static final void reportWarning(int sqlCode,
                                       java.lang.String funcName,
                                       java.lang.String[] tokens)
                                  throws WrapperException
```

Parameters

Table 224. Parameters for the reportWarning method

Name	Description
sqlcode	Predefined SQL code of the error that is reported.
funcName	Name of the function that reports the error. The string value cannot be greater than five characters. The client program can access this string value through the SQLERRP field of the SQLCA. The string value is in uppercase letters with a prefix of SQL.
tokens	Substitution tokens for the message.

Return value

None.

Throws

A WrapperException object if the operation fails.

traceEnabled method

Purpose

Verify whether DB2 Universal Database tracing is enabled.

Syntax

```
public static final boolean traceEnabled()
```

Parameters

None.

Return value

A value of true if tracing is enabled. Otherwise, the value is false.

Throws

None.

traceError method

Purpose

Trace an error message.

Syntax

```
public static final void traceError(int funcID,
                                    java.lang.String funcName,
                                    int probe,
                                    java.lang.String errorData)
```

Parameters

Table 225. Parameters for the traceError method

Name	Description
funcID	Trace ID of the caller function.
funcName	Name of the caller function.
probe	Probe ID.
errorData	Error data to trace.

Return value

None.

Throws

None.

traceException method

Purpose

Trace an exception and its call stack.

Syntax

```
public static final void traceException(int funcID,
                                        java.lang.String funcName,
                                        int probe,
                                        java.lang.Throwable exception)
```

Parameters

Table 226. Parameters for the traceException method

Name	Description
funcID	Trace ID of the caller function.
funcName	Name of the caller function.
probe	Probe ID.
exception	Exception to trace.

Return value

None.

Throws

None.

traceFunctionData method

Purpose

Trace function data using a single data trace parameter.

Syntax

```
public static final void traceFunctionData(int funcID,
                                           java.lang.String funcName,
                                           int probe,
                                           java.lang.String data)
```

Parameters

Table 227. Parameters for the traceFunctionData method

Name	Description
funcID	Trace ID of the caller function.
funcName	Name of the caller function.
probe	Probe ID.
data	Data to trace.

Return value
None.

Throws
None.

traceFunctionData method

Purpose
Trace function data using two data trace parameters.

Syntax

```
public static final void traceFunctionData(int funcID,
                                           java.lang.String funcName,
                                           int probe,
                                           java.lang.String data1,
                                           java.lang.String data2)
```

Parameters

Table 228. Parameters for the traceFunctionData method

Name	Description
funcID	Trace ID of the caller function.
funcName	Name of the caller function.
probe	Probe ID.
data1	The first set of data to trace.
data2	The second set of data to trace.

Return value
None.

Throws
None.

traceFunctionData method

Purpose
Trace function data using three data trace parameters.

Syntax

```
public static final void traceFunctionData(int funcID,
                                           java.lang.String funcName,
                                           int probe,
                                           java.lang.String data1,
                                           java.lang.String data2,
                                           java.lang.String data3)
```

Parameters

Table 229. Parameters for the traceFunctionData method

Name	Description
funcID	Trace ID of the caller function.
funcName	Name of the caller function.
probe	Probe ID.
data1	The first set of data to trace.
data2	The second set of data to trace.
data3	The third set of data to trace.

Return value

None.

Throws

None.

traceFunctionEntry method

Purpose

Trace a function entry.

Syntax

```
public static final void traceFunctionEntry(int funcID,
                                            java.lang.String funcName)
```

Parameters

Table 230. Parameters for the traceFunctionEntry method

Name	Description
funcID	Trace ID of the caller function.
funcName	Name of the caller function.

Return value

None.

Throws

None.

traceFunctionReturnCode method

Purpose

Trace a function return code.

Syntax

```
public static final void traceFunctionReturnCode(int funcID,
                                                 java.lang.String funcName,
                                                 int returnCode)
```

Parameters

Table 231. Parameters for the traceFunctionReturnCode method

Name	Description
funcID	Trace ID of the caller function.
funcName	Name of the caller function.
returnCode	Return code.

Return value

None.

Throws

None.

Related concepts:

- "Wrapper trace facility" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*

Related reference:

- "Wrapper utilities class" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*
- "WrapperException class (Java)" on page 190

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. The following list specifies the major accessibility features in DB2® Version 8 products:

- All DB2 functionality is available using the keyboard for navigation instead of the mouse. For more information, see "Keyboard input and navigation."
- You can customize the size and color of the fonts on DB2 user interfaces. For more information, see "Accessible display."
- DB2 products support accessibility applications that use the Java™ Accessibility API. For more information, see "Compatibility with assistive technologies" on page 208.
- DB2 documentation is provided in an accessible format. For more information, see "Accessible documentation" on page 208.

Keyboard input and navigation

Keyboard input

You can operate the DB2 tools using only the keyboard. You can use keys or key combinations to perform operations that can also be done using a mouse. Standard operating system keystrokes are used for standard operating system operations.

For more information about using keys or key combinations to perform operations, see Keyboard shortcuts and accelerators: Common GUI help.

Keyboard navigation

You can navigate the DB2 tools user interface using keys or key combinations.

For more information about using keys or key combinations to navigate the DB2 Tools, see Keyboard shortcuts and accelerators: Common GUI help.

Keyboard focus

In UNIX® operating systems, the area of the active window where your keystrokes will have an effect is highlighted.

Accessible display

The DB2 tools have features that improve accessibility for users with low vision or other visual impairments. These accessibility enhancements include support for customizable font properties.

Font settings

You can select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

For more information about specifying font settings, see Changing the fonts for menus and text: Common GUI help.

Non-dependence on color

You do not need to distinguish between colors in order to use any of the functions in this product.

Compatibility with assistive technologies

The DB2 tools interfaces support the Java Accessibility API, which enables you to use screen readers and other assistive technologies with DB2 products.

Accessible documentation

Documentation for DB2 is provided in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

Syntax diagrams are provided in dotted decimal format. This format is available only if you are accessing the online documentation using a screen-reader.

Related concepts:

- "Dotted decimal syntax diagrams" in the *Infrastructure Topics (DB2 Common Files)*

Related tasks:

- "Keyboard shortcuts and accelerators: Common GUI help"
- "Changing the fonts for menus and text: Common GUI help"

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

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

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _*enter the year or years*_. All rights reserved.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
DB2

The following terms are trademarks or registered trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

Index

N

O

P

Q

R

S

U

W

Contacting IBM

To contact IBM customer service in the United States or Canada, call 1-800-IBM-SERV (1-800-426-7378).

To learn about available service options, call one of the following numbers:

- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

To locate an IBM office in your country or region, see the IBM Directory of Worldwide Contacts on the Web at www.ibm.com/planetwide.

Product information

Information about DB2 Information Integrator is available by telephone or on the Web.

If you live in the United States, you can call one of the following numbers:

- To order products or to obtain general information: 1-800-IBM-CALL (1-800-426-2255)
- To order publications: 1-800-879-2755

On the Web, go to www.ibm.com/software/data/integration/db2ii/support.html. This site contains the latest information about:

- The technical library
- Ordering books
- Client downloads
- Newsgroups
- Fix packs
- News
- Links to Web resources

Comments on the documentation

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Information Integrator documentation. You can use any of the following methods to provide comments:

- Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Printed in USA

SC18-9173-00

Spine information:

IBM DB2 Information Integrator Java API Reference for Developing Wrappers Version 8.2