Java™ Foundation Class Reference

iPlanet™ Application Server

Version 6.0

8xx-xxxx-xx
January 2001

Contents

Preface

The *iPlanet Application Server Foundation Class Reference (Java)* provides specification-level documentation for the public classes and interfaces, and their methods, in the iPlanet Application Server Foundation Class Library. Use this book to look up how a particular class or interface method works, what syntax is required, and for examples on how to use it.

For conceptual and task-oriented information on designing and developing iPlanet Application Server applications, read the *Programmer's Guide (Java)*.

This preface contains the following sections:

- Using the Documentation
- How This Guide Is Organized
- Documentation Conventions
- Naming Conventions

Using the Documentation

The following table lists the tasks and concepts that are described in the iPlanet Application Server and iPlanet Application Builder printed manuals and online *Release Notes*. If you are trying to accomplish a specific task or learn more about a specific concept, refer to the appropriate manual.

Note that the printed manuals are also available as online files in PDF and HTML format.

Note that the printed manuals are also available online in PDF and HTML format, at: http://iplanet.com/manuals/ias.

For information about	See the following	Shipped with
Late-breaking information about the software and the documentation	*Release Notes*	iPlanet Application Server 6.0, iPlanet Application Builder 6.0

For information about	See the following	Shipped with
Installing iPlanet Application Server and its various components (Web Connector plug-in, iPlanet Application Server Administrator), and configuring the sample applications	*Installation Guide*	iPlanet Application Server 6.0
Installing iPlanet Application Builder.	`install.htm`	iPlanet Application Builder 6.0
Basic features of iPlanet Application Server, such as its software components, general capabilities, and system architecture.	*Getting Started Guide*	Available online.
Administering one or more application servers using the iPlanet Application Server Administrator Tool to perform the following tasks: • Deploying applications with the Deployment Manager tool • Monitoring and logging server activity • Setting up users and groups • Administering database connectivity • Administering transactions • Load balancing servers • Managing distributed data synchronization	*Administration & Deployment Guide*	iPlanet Application Server 6.0
Migrating your applications to the new iPlanet Application Server 6.0 programming model from the Netscape Application Server version 2.1, including a sample migration of an Online Bank application provided with iPlanet Application Server	*Migration Guide*	iPlanet Application Server 6.0, iPlanet Application Builder 6.0

For information about	See the following	Shipped with
Creating iPlanet Application Server 6.0 applications within an integrated development environment by performing the following tasks: • Creating and managing projects • Using wizards • Creating data-access logic • Creating presentation logic and layout • Creating business logic • Compiling, testing, and debugging applications • Deploying and downloading applications • Working with source control • Using third-party tools	*User's Guide*	iPlanet Application Builder 6.0
Creating iPlanet Application Server 6.0 applications that follow the open Java standards model (Servlets, EJBs, JSPs, and JDBC), by performing the following tasks: • Creating the presentation and execution layers of an application • Placing discrete pieces of business logic and entities into Enterprise Java Bean (EJB) components • Using JDBC to communicate with databases • Using iterative testing, debugging, and application fine-tuning procedures to generate applications that execute correctly and quickly	*Programmer's Guide (Java)*	iPlanet Application Builder 6.0
Using the public classes and interfaces, and their methods in the iPlanet Application Server class library to write Java applications	*Server Foundation Class Reference (Java)*	iPlanet Application Builder 6.0

For information about	See the following	Shipped with
Using the public classes and interfaces, and their methods in the iPlanet Application Server class library to write C++ applications	*Server Foundation Class Reference (C++)*	Order separately

How This Guide Is Organized

This guide is organized into three chapters and an appendix, as follows:

- Chapter 1, “About the Application Server Foundation Class Library”
- Chapter 2, “Classes”
- Chapter 3, “Interfaces”
- Appendix A, “Return Codes”

Documentation Conventions

File and directory paths are given in Windows format (with backslashes separating directory names). For Unix versions, the directory paths are the same, except that slashes are used instead of backslashes to separate directories.

This guide uses URLs of the form:

http://server.domain/path/file.html

In these URLs, *server* is the name of server on which you run your application; *domain* is your Internet domain name; *path* is the directory structure on the server; and *file* is an individual filename. Italic items in URLs are placeholders.

This guide uses the following font conventions:

- The `monospace` font is used for sample code and code listings, API and language elements (such as function names and class names), file names, pathnames, directory names, and HTML tags.
- *Italic* type is used for book titles, emphasis, variables and placeholders, and words used in the literal sense.

Naming Conventions

This guide uses the Java naming conventions detailed in the following table.

Item	Convention
Package name	Lowercase with periods indicating directory levels. For example, com.iPlanet.server.servlet.extension.
Class name	Mixed case with initial uppercase. For example, AppLogic class.
Interface name	Mixed case with initial uppercase. For example, IAppEventObj.
Method name	Mixed case with initial lowercase. For example, getTables( ).
Parameters	Mixed case with initial lowercase. For example, myQuery.

Chapter 1

About the Application Server Foundation Class Library

This chapter gives an overview of the classes and interfaces that make up the iPlanet Application Server Foundation Class Library.

This chapter includes the following main topics:

- Support for Standard Java APIs
- How to Use the iPlanet Application Server API

Support for Standard Java APIs

iPlanet Application Server supports several industry-standard Java APIs. In particular, iPlanet Application Server supports the APIs and technologies as defined by the following specifications:

- Java Servlet 2.1 API Specification
- Enterprise JavaBeans 1.0 Specification
- JavaServer Pages 0.92 Specification
- JDBC 2.0 Core API Specification
- JDBC 2.0 Standard Extensions Specification

Note that iPlanet Application Server 6.0 provides full support for JDBC 1.0 and partial support for JDBC 2.0. The supported JDBC 2.0 functionality is described in the *Programmer's Guide (Java)*.

The specifications listed above are accessible from the following location:

installdir`/iPlanet Application Server/docs/index.htm`

where *installdir* is the location in which you installed iPlanet Application Server.

Note that Sun Microsystems owns the process of defining, creating, and publishing these specifications. As a result, you can find more information about these APIs at their web site:

`http://java.sun.com`

This web site also hosts the latest versions of the Java specifications, which are not necessarily supported in iPlanet Application Server 6.0.

How to Use the iPlanet Application Server API

This guide, *iPlanet Application Server Foundation Class Reference (Java)*, provides information on the iPlanet Application Server API. If you want to develop an application that is 100% compliant with the industry-standard Java technologies (servlets, JSPs, EJBs, and JDBC calls), do not use the iPlanet Application Server API.

However, there are three main situations for which the iPlanet Application Server API is useful:

- To take advantage of additional iPlanet Application Server features (see: Additional iPlanet Application Server Features)
- To use the development environment provided by iPlanet Application Builder. The iPlanet Application Server API is used in some of the code that iPlanet Application Builder automatically generates (see: The Development Environment of iPlanet Application Builder)
- To use deprecated functionality for backward compatibility (see: Deprecated Functionality)

Additional iPlanet Application Server Features

In many situations, you'd like your application to use functionality that is not supported by the industry-standard APIs (for example, application events, security, and distributed state).

In addition, your application may need to access code such as:

- code that was built to run on a previous release of iPlanet Application Server
- other proprietary code

For these situations, you may want to use the features listed in the following table:

Classes	Interfaces	
GXContext	HttpServletRequest2	IEnumObject
iPlanet Application ServerRowSet	HttpSession2	ILock
SqlUtil	IAppEventMgr	IMailbox
	IAppEventObj	IServerContext
	ICallerContext	IState2
	IContext	

The Development Environment of iPlanet Application Builder

If you develop applications using iPlanet Application Builder, your generated code may rely on the classes or interfaces listed in the following table. Typically, only iPlanet Application Builder users use these classes or interfaces, although they are available to anyone developing applications in the iPlanet Application Server environment.

Classes	Interfaces
BaseUtils	IListRowSet
DBRowSet	IRowSet2
DefaultHttpSession	IServletErrorHandler
DefaultHttpSession2	
DefaultTemplateMap	
MemRowSet	
iPlanet Application ServerString	
iPlanet Application ServerStringBuffer	

Deprecated Functionality

iPlanet Application Server now supports a standards-based application model. As a result, many of the classes and interfaces in the iPlanet Application Server Foundation Class Library are deprecated. They are provided only for backward compatibility with existing iPlanet Application Server 6.0 applications.

Deprecated code will become unsupported in an upcoming release. iPlanet Application Server therefore recommends that new applications use equivalent functionality from the industry-standard Java APIs. In many cases, you may want to migrate existing applications, so that they operate within the new application model. For more information, see the *Migration Guide*.

The following classes and interfaces of the iPlanet Application Server API are deprecated. Within this reference guide, the corresponding headings are marked *(deprecated)*.

Classes	Interfaces	
AppLogic	IAppEvent	IResultSet
GUID	IBuffer	ISequence
GX	ICallableStmt	ISequenceMgr
GXVAL	IColumn	ISession2
Session2	IDataConn	ISessionIDGen
TemplateDataBasic	IDataConnSet	IStreamBuffer
TemplateMapBasic	IError	ITable
Util	IHierQuery	ITemplateData
	IHierResultSet	ITemplateMap
	IObject	ITile
	IOrder	ITrans
	IPreparedQuery	IValList
	IQuery	

Chapter 2

Classes

This chapter provides reference material on the classes in the iPlanet Application Server Foundation Class Library.

The following classes are included in this chapter:

AppLogic class *(deprecated)*

The AppLogic class is deprecated and is provided for backward compatibility only. AppLogics are not part of iPlanet Application Server's new application model, which is based on standards defined in the servlet, EJB, and JDBC specifications. The AppLogic class is provided so that existing AppLogics continue to be supported. However, new applications should avoid calling AppLogic methods whenever possible.

For information about replacing AppLogic functionality in existing applications, see the *Migration Guide*.

The AppLogic class is the base class for all AppLogic code. It provides a suite of useful AppLogic-related helper methods and member variables. You can, for example, use methods in your derived AppLogic class to create database connections, queries, transactions, and HTML output.

To use the AppLogic class, you must first import the com.kivasoft.applogic package at the beginning of your AppLogic file, as shown in the following example:

```
import com.kivasoft.applogic.*;
```

After you import the com.kivasoft.applogic package, you can create an instance of AppLogic and override the execute( ) method with AppLogic-specific code, as shown in the following example:

```
public class myAppLogic extends AppLogic {
   public int execute() {
   /* Override execute() method with AppLogic code */
      return result("Hello, world!\n");
   }
}
```

Package

com.kivasoft.applogic

Variables

Variable	Description
context	IContext object, which provides access to iPlanet Application Server services. Some objects require services fromIContext.
valIn	IValList object containing input parameters and other information. During the execute( ) method, an AppLogic can access items in the IValList to retrieve the arguments passed into the request.
valOut	IValList object containing output parameters. During the execute( ) method, the AppLogic can add or update items in the IValList to specify output values for the request.

Methods

Most methods of the AppLogic class are deprecated. Instead of calling these methods, we recommend using equivalent functionality as described with each AppLogic method.

In some cases, you may want to call AppLogic methods to access iPlanet Application Server features that are not currently available through the Java standards. For example, you may want to take advantage of caching or distributed state trees. In the following table, **boldface** type indicates a method that may be useful in iPlanet Application Server 6.0 applications.

Method	Description
createDataConn( )	Creates a new data connection object and opens a connection to a database or data source.
createDataConnSet( )	Creates a collection used to dynamically assign query name / data connection pairs before loading a query file.
createHierQuery( )	Creates a new query object used for building and running a hierarchical query.
createMailbox( )	Creates an electronic mailbox object used for communicating with a user through email.
createQuery( )	Creates a new query object used for building and running a flat query.
createSession( )	Creates a new session object used for tracking a user session.
createTrans( )	Creates a new transaction object used for transaction processing operations on a database.
deleteCache( )	Deletes the result cache for a specified AppLogic.
destroySession( )	Deletes a user session.
evalOutput( )	Creates an output report by merging data with a report template file.
evalTemplate( )	Creates an output report by merging data with a report template file. The report is an HTML document that can be viewed using a Web browser.
execute( )	Performs the main task of an AppLogic object, such as accessing a database, generating a report, or other operations. Should be overridden or implemented.
getAppEvent( )	Retrieves the application event object.

Method	Description
getSession( )	Returns an existing user session.
getStateTreeRoot( )	Returns an existing root node of a state tree or creates a new one.
isAuthorized( )	Checks a user's permission level to a specified action or AppLogic.
isCached( )	Returns true if AppLogic results are being saved in the result cache.
loadHierQuery( )	Creates a hierarchical query by loading a query file and one or more query names with associated data connections.
log( )	Writes a message to the server log.
loginSession( )	Logs an authorized user into a session with a secured application.
logoutSession( )	Logs a user out of a session with a secured application.
newRequest( )	Calls another AppLogic from within the current AppLogic.
newRequestAsync( )	Calls another AppLogic from within the current AppLogic, and runs it asynchronously.
removeAllCachedResults( )	Clears an AppLogic's result cache.
removeCachedResult( )	Clears specific results from an AppLogic's result cache.
result( )	Specifies the return value of an AppLogic.
saveSession( )	Saves changes to a session.
setCacheCriteria( )	Stores AppLogic results, such as HTML, data values, and streamed results, in a result cache.
setSessionVisibility( )	Sets the session visibility.
setVariable( )	Sets a value that is passed to later AppLogic requests that are called by the same client session.
skipCache( )	Skips result caching for the current AppLogic execution.
streamResult( )	Streams output results as a string.
streamResultBinary( )	Streams output binary data, such as a GIF file.
streamResultHeader( )	Streams output header data.

Example

The following example shows importing the com.kivasoft and Java packages, and overriding the execute( ) method with code that logs a series of messages:

```
import java.lang.*;
import com.kivasoft.*;
import com.kivasoft.types.*;
import com,kivasoft.util.*;
import com.kivasoft.applogic.*;

public class mySimpleAppLogic extends AppLogic {
   public int execute() {
      log("The execute method got called.");
      log("I can do anything I want here, such as");
      log("run queries, generate reports, update tables,");
      log("and log messages like this one.");
      return GXE.SUCCESS;
   }
}
```

createDataConn( )

Creates a new data connection object and opens a connection to a database or data source.

createDataConn( ) is deprecated. In standard-based applications, use the getConnection( ) method in the javax.sql.DataSource interface.

Syntax 1

Use this version for most database drivers.

```
public IDataConn createDataConn(
   int flags,
   int driver,
   String datasource,
   String database,
   String username,
   String password)
```

Syntax 2

Use this version for database drivers requiring parameters not found in Syntax 1. Provides parameters through an IValList object instead. To use this syntax, you must first create an instance of the IValList interface and use setValString( ) to specify the connection parameter names and values.

```
public IDataConn createDataConn(
   int flags,
   int driver,
   IValList prop,
   IContext context)
```

flags. One or more optional flags used for connecting to the specified data source.

- To try to use a cached connection, if one is available, specify GX_DA_CONN_FLAGS.GX_DA_CACHED. If no cached connections are currently available, a new one is created.
- To always create a new connection (instead of using a cached connection), specify GX_DA_CONN_FLAGS.GX_DA_NEW.
- To retry if a connection is not available, specify GX_DA_CONN_FLAGS.GX_DA_CONN_BLOCK.
- To return a failure immediately after the first attempt if a connection is not available, specify GX_DA_CONN_FLAGS.GX_DA_CONN_NOBLOCK.

The AppLogic can pass one parameter from both mutually exclusive pairs, as shown in the following example:

(GX_DA_CONN_FLAGS.GX_DA_CACHED |
GX_DA_CONN_FLAGS.GX_DA_CONN_BLOCK)

Specify 0 (zero) to use the system's default settings:
GX_DA_CONN_FLAGS.GX_DA_CACHED and
GX_DA_CONN_FLAGS.GX_DA_CONN_BLOCK

driver. Specify one of the following static variables in the GX_DA_DAD_DRIVERS class:

GX_DA_DRIVER_ODBC	GX_DA_DRIVER_SYBASE_CTLIB
GX_DA_DRIVER_MICROSOFT_JET	GX_DA_DRIVER_MICROSOFT_SQL
GX_DA_DRIVER_INFORMIX_SQLNET	GX_DA_DRIVER_INFORMIX_CLI
GX_DA_DRIVER_INFORMIX_CORBA	GX_DA_DRIVER_DB2_CLI
GX_DA_DRIVER_ORACLE_OCI	GX_DA_DRIVER_DEFAULT

If GX_DA_DRIVER_DEFAULT is specified, the iPlanet Application Server evaluates the drivers and their associated priorities set in the registry to determine the driver to use. Specify GX_DA_DRIVER_DEFAULT if your system uses ODBC and native drivers, and if you want the iPlanet Application Server to choose between an ODBC driver and a native driver at connection time.

datasource. Name of the data source to connect to. Used for databases that organize data into data source, database, and table objects and sub-objects. For example, a data source for Accounting might contain separate databases for GL, AP, and AR. Each database, such as Accounts Payable, might be a collection of individual tables, such as a vendor table, purchase order table, materials table, and so on. See your database server documentation for more information.

database. Name of the database to connect to. Used for database servers that organize data into databases and tables. See your database server documentation for more information.

username. Login user name that is valid for the specified database.

password. Login password that is valid for the specified user name and database.

props. IValList of connection-specific information required to log in to the data source. Use the following keys for the connection parameters:

- `"DSN"` for the data source name.
- `"DB"` for the database name.
- `"USER"` for the user name.
- `"PSWD"` for the password.

context. A pointer to the IContext object, which provides access to iPlanet Application Server services. Specify null.

Usage

A data connection is a communication link or session with a database or other data source. Before interacting with a data source, an AppLogic must first establish a connection with it. Each connection is represented by a data connection object, which contains all the information needed to communicate with a database or data source, such as the name of the database, database driver, user name, password, and so on. A data connection object is an instance of the IDataConn interface.

Use createDataConn( ) to set up a separate connection for each database or data source you want to access. AppLogic objects refer to the data connection object in their methods that perform subsequent operations on the database.

Rules

- Call createDataConn( ) before running any other database operations requiring a data connection object.

- Your network and the database server must be correctly configured and running so that the AppLogic on your application server can log into the database management system with which it will communicate.
- The data source name, database name, user name, and password must be valid for the database management system to which you want to connect.
- The AppLogic must log in with sufficient access rights to perform all operations it attempts on the data source.

Tips

- Before logging in to the database, the AppLogic should check the user's security level to verify sufficient access rights to perform intended operations on the database.
- The Data Access Engine (DAE) manages database connections and related housekeeping tasks, such as shutdown and cleanup. While the DAE performs these tasks automatically and intermittently, an AppLogic can also explicitly close data connections using closeConn( ) in the IDataConn interface.
- Before using an ODBC connection, you must use the ODBC administration utility supplied with your database software to define and name a data source. For more information about how to do this, refer to your ODBC documentation.
- To connect to a Sybase database, specify nullfor the datasource, and specify the database in the form of *server*:*database_name*. For example:

```
devds003:dnet00a
```

- A connection object can only be used by one thread at a time. When you need to access a database, get the connection object and save it in a local variable. When you're done, close the connection. For example:

```
Connection conn = DataSource.getConnection();

// code for database access

conn.close();
```

Return Value

IDataConn object representing the specified data connection, or null if a failure occurred. AppLogic uses the data connection object to uniquely identify this data connection in subsequent operations, such as queries and record insert, update, and delete operations.

Example

```
// Method to open a connection to a database
protected com.kivasoft.IDataConn getOBDataConn()
   {
      String username = "kdemo";
      String password = "kdemo";
      IDataConn newDataConn = createDataConn(0,
              GX_DA_DAD_DRIVERS.GX_DA_DRIVER_ODBC,
              /* Datasource name. */ "ksample",
              /* Database name.   */ "",
              /* userName.        */ username,
              /* password.        */ password);

      if (newDataConn == null)
      {
         log("ERROR: Could not create database connection");
      }
         return newDataConn;
   }
```

Related Topics

IDataConn interface,
IValList interface

createDataConnSet( )

Creates a collection used to dynamically assign query name / data connection pairs before loading a query file.

createDataConnSet( ) is deprecated. In standard-based applications, use the JDBC API to provide similar functionality.

Syntax

```
public IDataConnSet createDataConnSet(
   int flags)
```

flags. Specify 0. Internal use only.

Usage

Use createDataConnSet( ) only if you are loading a query file using loadHierQuery( ). To use a query file, an AppLogic first establishes a data connection with each database on which any queries will be run.

Next, the AppLogic calls createDataConnSet( ) to create an IDataConnSet object, then populates this collection with query name / data connection pairs. Each query name in the collection matches a named query in the query file. IDataConnSet provides a method for adding query name / data connection pairs to the collection. In this way, AppLogic can use standardized queries and select and assign data connections dynamically at runtime.

Finally, the AppLogic calls loadHierQuery( ) to create the hierarchical query object.

Return Value

IDataConnSet object that can hold a collection of data connections, or null for failure (such as insufficient memory).

Example

```
IDataConnSet connSet;
connSet = createDataConnSet(0);
// Specify query / db connection pairs
connSet.addConn("employee", conn_empDB);
connSet.addConn("sales", conn_salesDB);
IHierQuery hqry;
// Load the GXQ file with the db connection set
hqry = loadHierQuery("employeeReport.gxq",connSet, 0, null);

// Run the report
evalTemplate("employeeReport.html", hqry);
```

Related Topics

loadHierQuery( ),
IDataConnSet interface

createHierQuery( )

Creates a new query object used for building and running a hierarchical query.

createHierQuery( ) is deprecated. In standard-based applications, use the JDBC API to create join statements.

Syntax

```
public IHierQuery createHierQuery()
```

Usage

Use createHierQuery( ) for nested output or for merging query results with a template using evalOutput( ) or evalTemplate( ).

A hierarchical query can be more complex than a flat query. A hierarchical query combines one or more flat queries which, when run on the database server, returns a result set with multiple nested levels of data. The number of nested levels is limited only by system resources.

The hierarchical query is not necessarily a single query. In fact, a hierarchical query is a collection of one or more flat queries arranged in a series of cascading parent-child, one-to-many relationships. The parent query obtains the outer level of information, or summary, and the child query obtains the inner level of information, or detail. The parent level of information determines the grouping of information in its child levels. The child query is run multiple times, once for each row in the parent query's result set.

Tips

- Use createQuery( ) instead for simple, flat queries requiring tabular, non-nested output that is merged with HTML templates.
- To use a hierarchical query, an AppLogic first creates each individual flat query and defines its selection criteria. Next, it creates the IHierQuery object with createHierQuery( ), then calls addQuery( ) repeatedly to add a child query to a parent query for each level of detail in the hierarchical query.
- Alternatively, an AppLogic can create a hierarchical query by loading a query file using loadHierQuery( ). With this technique, the iPlanet Application Server can cache query objects to service requests for identical queries more quickly.

Return Value

IHierQuery object representing a hierarchical query, or null for failure. An AppLogic module uses this object to uniquely identify this hierarchical query in subsequent operations, such as defining the query criteria, executing the query, retrieving query results, and processing templates.

Example 1

```
// Create the flat query

IQuery qry = createQuery();

qry.setTables("CTLusers");

qry.setFields("loginName, Password, AccessLevel")

qry.setOrderBy("LoginName);
```

```
// Create the hierarchical query used for template processing
IHierQuery hqry = createHierQuery();

// Add the flat query object and data connection to hqry
hqry.addQuery(qry, conn, "USERS", "", "");

// Pass hierarchical query to evalTemplate() for reporting
if(evalTemplate("apps/template/userinfo.html", hqry)== GXE.SUCCESS)
   return result("");
else
   return result("Failed to Generate HTML");
}
```

Example 2

```
// Create the flat query
IQuery qry = createQuery();
qry.setTables("CTLusers");
qry.setFields("loginName, Password, AccessLevel")
qry.setWhere("UserId > 100");

// Create the hierarchical query
IHierQuery hqry = createHierQuery();
hqry.addQuery(qry, conn, "USERS", "", "");

// Execute the hierarchical query
IHierResultSet hrs = hqry.execute(0, 0, null);

// Process rows in result set
if(hrs.getRowNumber("USERS")!=0)
. . .
```

Related Topics

addQuery( ) in the IHierQuery interface,
createDataConn( ),
createQuery( ),
execute( ) in the IHierQuery interface,
IHierQuery interface,
IHierResultSet interface

createMailbox( )

Creates an electronic mailbox object used for communicating with a user's mailbox.

Syntax

```
public IMailbox createMailbox(
    String pHost,
    String pUser,
    String pPassword,
    String pUserAddr)
```

pHost. Address of POP and SMTP server, such as mail.myOrg.com. If the POP and SMTP servers are running on different hosts, you must use two separate createMailbox( ) calls.

pUser. Name of user's POP account, such as jdoe.

pPassword. Password for POP server.

pUserAddr. Return address for outgoing mail, such as john@myOrg.com. Usually the electronic mail address of the user sending the message.

Usage

Use createMailbox( ) to set up a mail session for sending and receiving electronic mail messages.

In the Internet electronic mail architecture, different servers are used for incoming and outgoing messages.

- POP (post-office protocol) servers process incoming mail and forward messages to the recipient's mailbox.
- SMTP (simple mail transport protocol) servers forward outgoing mail to the addressee's mail server.

Rules

- The specified user account and password must be valid for the specified POP host name.

- The user address must be valid for the specified SMTP server.

Tip

Once instantiated, use the methods in the IMailbox interface to open and close a mailbox, as well as send and receive mail messages.

Return Value

IMailbox object representing a mailbox, or null for failure (such as an invalid parameter).

Related Topics

IMailbox interface

createQuery( )

Creates a new query object used for building and running a flat query.

createQuery( ) is deprecated. In standard-based applications, use the createStatement( ) method in the java.sql.Connection interface.

Syntax

```
public IQuery createQuery()
```

Usage

A flat query is the simplest type of query. It retrieves data in a tabular, non-hierarchical result set. Unlike a hierarchical query, a flat query returns a result set that is *not* divided into levels or groups.

An AppLogic can also use createQuery( ) to create a query object to perform SELECT, INSERT, DELETE, or UPDATE operations on a database.

Tips

- To query a database, the AppLogic first uses createQuery( ) to create the query object, then constructs the query selection criteria using methods in the IQuery interface, and finally runs the query on a database server. The AppLogic can process results using methods in the IResultSet interface.
- Alternatively, AppLogic can pass a SQL SELECT statement directly to the database server using setSQL( ) in the IQuery interface.
- To retrieve data with nested levels of information, use createHierQuery( ) instead.

Return Value

IQuery object representing a query, or null for failure. AppLogic uses this object to uniquely identify this query in subsequent operations, such as defining the query criteria, executing the query, retrieving query results, and processing templates.

Example

```
// Create the flat query object
IQuery qry = createQuery();
// Set up the query
qry.setTables("CTLcust");
qry.setFields("CustomerID, Customer");
qry.setWhere("Customer"+"='"+String.valueOf(custId)+"'");

// Execute the query
IResultSet rs = conn.executeQuery(0, qry, null, null);

// Check for a result set with rows
if((rs!=null)&&(rs.getRowNumber()>0))
   return result("Sorry, this user ("+
   firstName+" "+lastName+") already exists");
// Otherwise, process the result set . . .
```

Related Topics

createDataConn( ),
createQuery( ),
IHierQuery interface,
IHierResultSet interface,
executeQuery( ) in the IDataConn interface

createSession( )

Creates a new session object used for tracking a user session.

createSession( ) is deprecated. In standard-based applications, use the getSession( ) method in the interface javax.servlet.http.HttpSession, and set the boolean parameter to true.

Syntax

```
public ISession2 createSession(
    int dwFlags,
    int dwTimeout,
    String pAppName,
    String pSessionID,
    ISessionIDGen pIDGen)
```

dwFlags. Specify one of the following flags, or 0 to use the default system settings:

- GXSESSION.GXSESSION_LOCAL to make the session visible to AppLogics in the local process only.
- GXSESSION.GXSESSION_CLUSTER to make the session visible to all AppLogics within the cluster.
- GXSESSION.GXSESSION_DISTRIB to make the session visible to all AppLogics on all iPlanet Application Servers.
- GXSESSION.GXSESSION_TIMEOUT_ABSOLUTE to specify that the session expires at a specific date and time. Do not use this flag. It is currently unimplemented but reserved for future use.
- GXSESSION.GXSESSION_TIMEOUT_CREATE to specify that the session expires *n* seconds from the time the session was created.

The default scope is distributed and the default timeout is 60 seconds from the time the session was last accessed.

dwTimeout. Session timeout, in number of seconds, or zero for no timeout. The meaning of timeout depends on the timeout flag specified in dwFlags. A value of 0 means the session is deleted when the AppLogic calls destroySession( ).

pAppName. Name of the application associated with the session. The application name enables the iPlanet Application Server to determine which AppLogics have access to the session data. Specify null to use the application name assigned to the AppLogic during kreg registration.

pSessionID. The session ID to use. Specify null to use the default ID generated by the system.

pIDGen. The session ID generation object used to generate session IDs. Specify null.

Usage

Use createSession( ) to create a new session between a user and your application. AppLogics use sessions to store information about each user's interaction with an application. For example, a login AppLogic might create a session object to store the user's login name and password. This session data is then available to other AppLogics in the application.

Rule

If you implement a custom session class, you must override createSession( ).

Return Value

ISession2 object representing a user session, or null for failure.

Example

In the following code, getSession( ) checks if a session exists. If there isn't an existing session, createSession( ) creates a new session.

```
ISession2 sess;
sess = getSession(0, "Catalog", null);
if (sess == null)
{
   log("Could not get session, creating a new one");
   sess = createSession(GXSESSION.GXSESSION_DISTRIB, 0,
                null, null, null);
```

Related Topics

getSession( ),
saveSession( ),
Session2 class,
ISession2 interface

createTrans( )

Creates a new transaction object used for transaction processing operations on a database.

createTrans( ) is deprecated. In standard-based applications, call setAutoCommit(FALSE), a method in the java.sql.Connection interface.

Syntax

```
public ITrans createTrans()
```

Usage

Transaction processing allows the AppLogic to define a series of operations that succeed or fail as a group. If all operations in the group succeed, then the system commits, or saves, all of the modifications from the operations. If any operation in the group fails for any reason, then the AppLogic can roll back, or abandon, any proposed changes to the target table(s).

If your application requires transaction processing, use createTrans( ) to create a transaction object. Pass this transaction object to subsequent methods, such as addRow( ) or executeQuery( ), that make up a transaction.

Tips

- Use this method in conjunction with addRow( ),updateRow( ), and deleteRow( ) methods in the ITable interface and executeQuery( )in the IDataConn interface.
- To manage transaction processing operations, use createTrans( ) to create an instance of the ITrans interface, then use begin( ), commit( ), and rollback( ) in the ITrans interface to begin, commit, and rollback the transaction, respectively.

Return Value

ITrans object representing a transaction, or null for failure. AppLogic uses this object to uniquely identify this transaction in subsequent transaction processing operations, such as beginning, committing, or rolling back a transaction.

Example

```
// Create and begin a transaction
ITrans trx = createTrans();
trx.begin();

// 1) Process the credit card
if(!processCreditCard(cusId, card, number, expirationDate, trx))
{
   trx.rollback();
   return result("Could not process the credit card
information"); }

// 2) Process the invoice record
InfoHolder info=new InfoHolder();
if(!makeInvoiceRecord(cusId, number, trx, info)) {
   trx.rollback();
   return result("Could not create the invoice record"); };

// 3) Process products on the invoice
if(!makeInvoiceEntries(info.invoiceId, trx)) {
```

```
   trx.rollback();
   return result("Can't create product records"); };

// 4) Process optional shipping information
if(shippingInfo && !makeShippingRecord(info.invoiceId, trx,
addr1, addr2, city, state, zip)) {
   trx.rollback();
   return result("Could not create the shipping information
record"); };

// 5) Process the inventory for each purchased product
if(!reduceProductInventory(trx)) {
   // Problem occurred - abandon everything
   trx.rollback();
    return result("Could not reduce inventory");
}

// No problem occurred - save everything
trx.commit(0);
// Return success message / report
```

Related Topics

ITrans interface

deleteCache( )

Deletes the result cache for a specified AppLogic.

Syntax

```
public int deleteCache(
   String guid)
```

guid. The guid that identifies the AppLogic whose result cache to delete. Specify null to delete the current AppLogic's cache.

Usage

To free system resources, use deleteCache( ) to clear all results from an AppLogic's cache when the results are no longer needed. This method also stops further caching of results.

Tips

- To clear an AppLogic's result cache, but continue caching, use removeAllCachedResults( ).
- To clear a specific result from the cache, use removeCachedResult( ).

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
int hr;
String guid = valIn.getValString("applogic");

hr = deleteCache(guid);

if (hr == GXE.SUCCESS)
   return result("Successfully deleted cache")
else
   return result("Failed to delete cache");
```

Related Topics

removeAllCachedResults( ),
removeCachedResult( ),
setCacheCriteria( )

destroySession( )

Deletes a user session.

destroySession( ) is deprecated. In standard-based applications, use the invalidate( ) method in the interface javax.servlet.http.HttpSession.

Syntax

```
public int destroySession(
   ISessionIDGen pIDGen)
```

pIDGen. The session ID generation object used to generate session IDs. Specify null .

Usage

To increase security and conserve system resources, use destroySession( ) to delete a session between a user and the application when the session is no longer required. An AppLogic typically calls destroySession( ) when the user logs out of an application.

Tip

If the AppLogic set a timeout value for the session when it was created, you need not delete the session explicitly with destroySession( ). The session is deleted automatically when the timeout expires.

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

createSession( ),
getSession( )

evalOutput( )

Creates an output report by merging data with a report template file. Depending on the client—AppLogic or web browser—evalOutput( ) returns either a self-describing data stream or HTML output.

evalOutput( ) is deprecated. In standard-based applications, similar functionality can be achieved by using the interface javax.servlet.RequestDispatcher.

Syntax 1

Merges a template with data from a hierarchical query object.

```
public int evalOutput(
   String templatePath,
   IHierQuery query,
   ITemplateMap map,
   IStream stream,
   IValList props);
```

Syntax 2

Merges a template with data from an ITemplateData object or IHierResultSet object. IHierResultSet objects implement the ITemplateData interface.

```
public int evalOutput(
   String templatePath,
   ITemplateData data,
   ITemplateMap map,
   IStream stream,
   IValList props);
```

templatePath. Path to the template file used to create the report. At a minimum, specify the file name. Do not specify the filename extension; for example, specify "report" instead of "report.html". The evalOutput( ) method automatically uses the correct filename extension depending on the client type. Use a relative path whenever possible. The iPlanet Application Server first searches for the template using the specified path. If the template is not found, the iPlanet Application Server uses the configured TEMPLATE\PATH search path to find it. For more information on configuring the search path, see the *Administration and Deployment Guide*.

query. Hierarchical query object from which evalOutput( )createTrans( ) derives the hierarchical result set to merge with the template. The Template Engine runs the query on the database server. To specify this parameter, the AppLogic must first create the specified hierarchical query, using createHierQuery( ) in the AppLogic class, and then define it using methods in the IHierQuery interface or calling loadHierQuery( ).

map. Field map that links template fields to calculated values. Fields in the template are expressed with the cell type gx tags. Additionally, the map can be used to map source data with a non-matching field name but identically-formatted data. To specify this parameter, the AppLogic should instantiate the TemplateMapBasic class, add template / field mappings using put( ) in the ITemplateMap interface, then pass the populated ITemplateMap object to evalOutput( ) for template processing.

stream. The output stream where results will be captured for subsequent retrieval and processing. Specify null to use the default stream, which sends results back to the client. To specify this parameter, an AppLogic creates a stream buffer object from IStreamBuffer, which it passes to evalOutput( ). After evalOutput( ) returns, the AppLogic calls getStreamData( ) in the IStreamBuffer interface to retrieve the contents of the buffer as an array of byte values.

data. ITemplateData object containing data. This can be a hierarchical result set from executing a hierarchical query or it can be data programmatically organized in memory. To specify this data in memory, an AppLogic must first instantiate the TemplateDataBasic class (or implement your own version of the ITemplateData interface), populate the ITemplateData object with rows of hierarchical data, then pass it to evalOutput( ) for template processing.

props. Specify null.

Usage

Use evalOutput( ) in an AppLogic that returns output to different types of clients. The evalOutput( ) method detects the client type, selects the appropriate template file to merge with the data, and generates the appropriate output, as described in the following table:

Client	Template File Used by evalOutput( )	Output Returned by evalOutput( )
Web browser	HTML	HTML page
AppLogic that passed to its newRequest( ) call the following key and value in the input IValList parameter: key: gx_client_type value: "ocl"	GXML	Self-describing data stream, which contains the names of the fields in the result set and their values.
AppLogic that does not specify a client type explicitly, or that passed to its newRequest( ) call the following key and value in the input IValList parameter: key: gx_client_type value: "http"	HTML	HTML page

Both the GXML and HTML template files contain embedded tags, called GX tags, that specify how the Template Engine merges dynamic data with the template to produce the output report. In addition, the HTML template file can contain graphics, static text, and other components, just like any HTML-formatted document.

The data that the Template Engine merges with the template can come from several sources. Most commonly, it comes from the result set of a hierarchical query. However, it can also come from an ITemplateData object containing data organized hierarchically in memory.

Tips

- If possible, write queries so that field names in the result set match the field names in the template. Otherwise, you must use an ITemplateMap object to map field names.

- To create an GXML file, you can convert an HTML template file with the khtml2gxml utility. This utility strips HTML tags from the template file and saves the file as a GXML file. The following is an example of how to run the utility from the command line:

  ```
  khtml2gxml mytemplate.html
  ```

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Create a hierarchical query used for template processing
IHierQuery hqry = createHierQuery();

// Add a flat query object and data connection to hqry
hqry.addQuery(qry, conn, "USERS", "", "");

// Pass hierarchical query to evalOutput() for reporting
if(evalOutput("apps/template/userinfo", hqry)== GXE.SUCCESS)
   return result("");
else
   return result("Failed to Generate HTML");
}
```

Related Topics

evalTemplate( ),
result( ),
IHierQuery interface,
TemplateDataBasic class and the ITemplateData interface,
TemplateMapBasic class and the ITemplateMap interface

evalTemplate( )

Creates an output report by merging data with a report template file. The report is an HTML document that can be viewed using a Web browser.

evalTemplate( ) is deprecated. In standard-based applications, similar functionality can be achieved by using the interface javax.servlet.RequestDispatcher.

Syntax 1

Merges an HTML report template with data from a hierarchical query object.

```
public int evalTemplate(
   String path,
   IHierQuery query)
```

Syntax 2

Merges an HTML report template with data from a hierarchical query object.

```
public int evalTemplate(
   String path,
   IHierQuery query,
   ITemplateMap map,
   IStream stream)
```

Syntax 3

Merges an HTML report template with data from an ITemplateData object or IHierResultSet object. IHierResultSet objects implement the ITemplateData interface.

```
public int evalTemplate(
   String path,
   ITemplateData data,
   ITemplateMap map)
```

path. Path to the HTML template file used to create the report. At a minimum, specify the file name. Use a relative path whenever possible. The iPlanet Application Server first searches for the template using the specified path. If the template is not found, the iPlanet Application Server uses the configured TEMPLATE\PATH search path to find it. For more information on configuring the search path, see the *Administration and Deployment Guide*.

query. Hierarchical query object from which evalTemplate( ) derives the hierarchical result set to merge with the HTML template. The Template Engine runs the query on the database server. To specify this parameter, the AppLogic must first create the specified hierarchical query, using createHierQuery( ) in the AppLogic class, and then define it using methods in the IHierQuery interface or calling loadHierQuery( ).

map. Field map that links template fields to calculated values. Fields in the template are expressed with the cell type gx tags. Additionally, the map can be used to map source data with a non-matching field name but identically-formatted data. To specify this parameter, the AppLogic should instantiate the TemplateMapBasic class, add template / field mappings using put( ) in the ITemplateMap interface, then pass the populated ITemplateMap object to evalTemplate( ) for template processing.

stream. The output stream where results will be captured for subsequent retrieval and processing. Specify null to use the default stream, which sends results back to the client. To specify this parameter, an AppLogic creates a stream buffer object from IStreamBuffer, which it passes to evalOutput( ). After evalTemplate( ) returns, the AppLogic calls getStreamData( ) in the IStreamBuffer interface to retrieve the contents of the buffer as an array of byte values.

data. ITemplateData object containing data. This can be a hierarchical result set from executing a hierarchical query or it can be data programmatically organized in memory. To specify this data in memory, an AppLogic must first instantiate the TemplateDataBasic class (or implement your own version of the ITemplateData interface), populate the ITemplateData object with rows of hierarchical data, then pass it to evalTemplate( ) for template processing.

Usage

Use evalTemplate( ) to create an HTML report by merging data with an HTML template file. An HTML template is an HTML document with the addition of special embedded tags, called GX tags, that specify how the Template Engine merges dynamic data with the template to produce the output report or HTML page. In addition to these dynamic links, a template can contain static text, graphics, and other components, just like any HTML-formatted document.

The data that the Template Engine merges with the template can come from several sources. Most commonly, it comes from the result set of a hierarchical query. However, it can also come from an ITemplateData object containing data organized hierarchically in memory.

Tips

- If your AppLogic requires the flexibility of returning different output depending on the client—a Web browser or another AppLogic—use evalOutput( ) instead.
- If possible, write queries so that field names in the result set match the field names in the template. Otherwise, you must use an ITemplateMap object to map field names.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Create the flat query
IQuery qry = createQuery();
qry.setTables("CTLusers");
```

```
qry.setFields("loginName, Password, AccessLevel");
qry.setOrderBy("LoginName");

// Create the hierarchical query used for template processing
IHierQuery hqry = createHierQuery();

// Add the flat query object and data connection to hqry
hqry.addQuery(qry, conn, "USERS", "", "");

// Pass hierarchical query to evalTemplate() for reporting
if(evalTemplate("apps/template/userinfo.html",
hqry)==GXE.SUCCESS)
   return result("");
else
   return result("Failed to Generate HTML");
}
```

Related Topics

evalOutput( ),
IHierQuery interface,
TemplateDataBasic class and the ITemplateData interface,
TemplateMapBasic class and the ITemplateMap interface

execute( )

Performs the main task of an AppLogic, such as accessing a database, generating a report, or other operations. It should be overridden in your AppLogic subclass.

execute( ) is deprecated. In standard-based applications, use one of the request processing methods in the class javax.servlet.http.HttpServlet. For example, use doGet( ), doPost( ), or service( ).

Syntax

```
public int execute()
```

Usage

iPlanet Application Server calls the AppLogic's execute( ) method automatically whenever a request is received for an AppLogic, such as when a user submits a form or an information request.

Rule

By default, execute( ) does nothing except return a value of zero (0). You should always write code to override this method in your AppLogic derived class.

Tips

- In general, your AppLogic class will inherit from the AppLogic class and override the default behavior of the execute( ) method, such as retrieving an orders report from a database.
- The AppLogic can analyze the valIn member variable for input arguments using methods in the IValList interface.
- The AppLogic can modify the valIn member variable using methods in the IValList interface.

Return Value

GXE.SUCCESS if the method succeeds.

Example

The following example overrides the execute( ) method to display “Hello world”:

```
package gxApp.sample

import java.lang.*;

import com.kivasoft.*;

import com.kivasoft.types.*;

import com,kivasoft.util.*;

import com.kivasoft.applogic.*;


public class HelloWorldAppLogic extends AppLogic {

public int execute() {

// Simple code that overrides execute() method

   return result("Hello world!");

   }

}
```

Related Topics

result( ),
IValList interface

getAppEvent( )

Retrieves the application event object.

getAppEvent( ) is deprecated. See New Usage section for more information.

Syntax

```
public IAppEvent getAppEvent()
```

New Usage

This method is deprecated and is provided for backward compatibility only.

New applications should use the IAppEventMgr Interface and IAppEvent Interface (deprecated), along with the GetAppEventMgr( ) method in the com.kivasoft.dlm.GXContext class.

Old Usage

Use getAppEvent( )to retrieve an IAppEvent object. Through the IAppEvent interface, you can create and manage application events. An AppLogic uses application event objects to define events that are triggered at a specified time or times or when triggered explicitly.

Return Value

IAppEvent object, or null for failure.

Related Topics

IAppEvent interface,
registerEvent( ) in the IAppEvent interface

getSession( )

Returns an existing user session.

getSession( ) is deprecated. In standard-based applications, use the getSession( ) method in the interface javax.servlet.http.HttpSession.

Syntax

```
public ISession2 getSession(
   int dwFlags,
   String appName,
   ISessionIDGen pIDGen)
```

dwFlags. Specify 0 (zero).

pAppName. Name of the application associated with the session. The application name enables the iPlanet Application Server to determine which AppLogics have access to the session data. Specify null to use the application name assigned to the AppLogic during kreg registration.

pIDGen. The session ID generation object used to generate session IDs. Specify null.

Usage

Use getSession( ) to obtain an existing session. Use it also to determine if a user session exists before calling createSession( ) to create one.

Rule

If you implement a custom session class, you must implement your own method to get a session, which in turn, can call the getSession( ) method.

Return Value

ISession2 object representing a user session, or null for failure.

Example 1

In the following code, getSession( ) checks if a session exists. If there isn't an existing session, createSession( ) creates a new session.

```
ISession2 sess;
sess = getSession(0, "Catalog", null);
if (sess == null)
{
   log("Could not get session, creating a new one");
   sess = createSession(GXSESSION.GXSESSION_DISTRIB, 0,
               null, null, null);
```

Example 2

In the following code, getSession( ) gets an existing session, then checks if the user is authorized to perform a secured task:

```
public class ProductRestock extends AppLogic
{
  public int execute()
    {
      try {
         ISession2 mySess;
         mySess = getSession(0, null, null);
         if (mySess == null) {
            log("missing session");
            return evalOutput("kivaapp/shop/please_login",
                  (ITemplateData) null,
                  (ITemplateMap)  null, null, null);
         }
         // Check to see if the current role is authorized
         // to update inventory.
         //
```

```
if (isAuthorized("Shop_Inventory", "WRITE") !=
  GXACLPERMSTATUS.GXACL_ALLOWED)
{
   log("unauthorized access: Shop_Inventory");
   return evalOutput("kivaapp/shop/no_access",
         (ITemplateData) null,
         (ITemplateMap)  null, null, null);
}
// Record the inventory restock transaction.
```

Related Topics

createSession( ),
loginSession( ),
saveSession( ),
Session2 class,
ISession2 interface

getStateTreeRoot( )

Returns an existing root node of a state tree or creates a new one.

Syntax

```
public IState2 getStateTreeRoot(
   int dwFlags,
   String pName)
```

dwFlags. Specify one of the following flags or zero to use the default settings:

- GXSTATE.GXSTATE_LOCAL to make the node visible to the local process only.
- GXSTATE.GXSTATE_CLUSTER to make the node visible to all AppLogics within the cluster.
- GXSTATE.GXSTATE_DISTRIB, the default, to make the node visible to all AppLogics on all servers.

pName. The name of the root node. If a node with this name doesn't already exist, a new node is created.

Usage

Use getStateTreeRoot( ) to create a state tree. A state tree is a hierarchical data storage mechanism. It is used primarily for storing application data that needs to be distributed across server processes and clusters.

Return Value

IState2 object representing the root node, or null for failure.

Example

The following code shows how to create a state tree and a child node:

```
IState2 tree = getStateTreeRoot(GXSTATE.GXSTATE_DISTRIB,
"Grammy");

if (tree!=null)
{
   IState2 child = tree.getStateChild("Best Female Vocal");
   if (child == null)
   {
      child = tree.createStateChild("Best Female Vocal", 0,
        GXSTATE.GXSTATE_DISTRIB);
```

Related Topics

IState2 interface

isAuthorized( )

Checks a user's permission level to a specified action or AppLogic.

isAuthorized( ) is deprecated. As of the 2.1 servlet specification, there is no standard security model for servlets. However, iPlanet Application Server provides a security model. In servlets, use the equivalent isAuthorized( ) method, available from the HttpSession2 Interface in package com.iplanet.server.servlet.extension.

Syntax 1

Use in most cases.

```
public int isAuthorized(
   String pTarget,
   String pPermission)
```

Syntax 2

Contains several parameters that are placeholders for future functionality.

```
public int isAuthorized(
   String pDomain,
   String pTarget,
   String pPermission,
   int method,
   int flags,
   ICred pCred,
   IObject pEnv)
```

pDomain. The type of Access Control Lists (ACL). An ACL (created by the server administrator) defines the type of operations, such as Read or Write, that a user or group can perform. There are two types of ACLs: AppLogic and general. For this parameter, specify one of the following strings, which specifies the type of ACL to check for this user:

"kiva:acl,logic"

"kiva:acl,general"

pTarget. The name of the ACL, if the ACL is a general type. If the ACL is an AppLogic ACL, specify the AppLogic name or GUID string.

pPermission. The type of permission, for example, “EXECUTE.”

method. Specify 0.

flags. Specify 0.

pCred. Specify null.

pEnv. Specify null.

Usage

Use isAuthorized( ) in portions of the code where application security is enforced through Access Control Lists (ACL). This method lets an application check if a user has permission to execute an AppLogic or perform a particular action. The application can use the result of isAuthorized( ) as a condition in an If statement. It can, for example, return a message to users who are denied access to an AppLogic.

Application developers should obtain the list of registered ACLs, users and groups from the server administrator who created these items. ACLs are created through the Enterprise Administrator tool or through the kreg tool.

Rule

Before calling isAuthorized( ), the application must create a session with createSession( ) and a user must be logged in with loginSession( ).

Return Value

One of the following:

Value	Description
GXACLPERMSTATUS.GXACL_ALLOWED	The specified permission is granted to the user.
GXACLPERMSTATUS.GXACL_NOTALLOWED	The specified permission is not granted to the user.
GXACLPERMSTATUS.GXACL_DONTKNOW	The specified permission is unlisted or there is conflicting information.

Example

```
if (isAuthorized("MonthlyForecast", "READ") !=
GXACLPERMSTATUS.GXACL_ALLOWED)
   return result("<html><body>sorry no access</body></html>);
else
// run monthly forecast report
```

Related Topics

loginSession( )

isCached( )

Returns true if AppLogic results are being saved in the result cache.

Syntax

```
public boolean isCached()
```

Usage

Call isCached( ) to determine whether caching is enabled for the current AppLogic. You should, for example, call isCached( ) before calling setCacheCriteria( ) to avoid inadvertently overwriting the current contents of the result cache.

Return Value

A boolean true if caching is enabled, or a boolean false if not.

Example

```
// Determine whether AppLogic is cached
// before setting cache criteria
if(!isCached()) {
   log("Setting agent cache criteria");
   if(setCacheCriteria(60, 3, "category")!=0)
      log("Could not set agent cache criteria");
   else
      log("Succeeded in setting agent cache criteria");
   }
else
      log("Not setting agent cache criteria");
```

Related Topics

skipCache( )

loadHierQuery( )

Creates a hierarchical query by loading a query file containing one or more query names and associated data connections.

loadHierQuery( ) is deprecated. In standard-based applications, use the JDBC API to execute a join statement.

Syntax

```
public IHierQuery loadHierQuery(
   String pFileName,
   IDataConnSet pDataConnSet,
   int flags,
   IValList pParams)
```

pFileName. Name of the query (.GXQ) file, including the path. Use a relative path when possible.

A query file is an ASCII text file containing one or more SQL statements. You can create the file using any ASCII text editor. Use the following syntactical guidelines:

- The file for a hierarchical query contains several SQL SELECT statements (compliant with ANSI SQL89) with the following additions:
  - Each query is preceded by the following line:

    `query` *`queryName`* `using (`*`driverCode`*`,` *`DSN`*`,` *`UserName`*`) is`
  - For a child query, append the following line after the SQL SELECT statement:

```
join currentQueryName to parent parentName where
currentQueryName.table.column = parentName.colorAlias
```

- In the query file, do not use any semicolons (;) or other vendor-specific SQL statement terminators.

pDataConnSet. Collection of query name / data connection pairs. The query names in the collection must match the named queries in the query file. The associated IDataConn object identifies the data connection for the query.

flags. Specify 0 (zero). Internal use only.

pParams. IValList of query file parameters, or null. A collection of placeholders for the WHERE clause. A placeholder may be a name or a number. It is prefixed by a colon (:) character. The placeholders can be replaced by specifying replacement values in the ValList parameter.

Usage

Use loadHierQuery( ) to create a hierarchical query object. An AppLogic can retrieve standardized queries stored in a data file and, at runtime, can dynamically select and assign the data sources on which the query is run. You create the query file separately using the Query Designer or an ASCII text editor, ANSI 89 standard SQL SELECT statements, and specialized syntax. A query file can define both flat and hierarchical queries.

To use a query file, the AppLogic first establishes a data connection with each database on which any queries will be run. Next, the AppLogic calls createDataConnSet( ) in the AppLogic class to create an IDataConnSet collection, then populates this collection with query name / data connection pairs. Each query name in the collection matches a named query in the query file.

IDataConnSet provides a method for adding query name / data connection pairs to the collection. In this way, AppLogic can use standardized queries and assign data connections dynamically at runtime. Finally, the AppLogic calls loadHierQuery( ) to create the hierarchical query object.

Rules

- AppLogic must first call createDataConnSet( ) to create an IDataConnSet, then add query name / data connection pairs using addConn( ) in the IDataConnSet interface.
- The query names in the collection must match the query names in the query file.

Return Value

IHierQuery object representing a hierarchical query, or null for failure (such as query file not found). The AppLogic uses this object to uniquely identify this hierarchical query in subsequent operations, such as defining the query criteria, executing the query, retrieving query results, and processing templates.

Example

The following example shows a query (GXQ) file and a section of an AppLogic that loads the hierarchical query file and creates an HTML report:

Query file:

```
/* STATES */
query STATES using (ODBC, kstates, kuser) is
select STATES.STATE as STATES_STATE
from STATES
where (STATES.REGION = ':REGION')
order by STATES.STATE asc

/* DETAILS */
query DETAILS using (ODBC, kdetails, kuser) is
select COUNTIES.COUNTYNAM as COUNTIES_COUNTYNAM,
   COUNTIES.POP as COUNTIES_POP,
   COUNTIES.STATE as COUNTIES_STATE
from COUNTIES
order by COUNTIES.COUNTYNAM asc

join DETAILS to parent STATES
where DETAILS.COUNTIES.STATE = 'STATES.STATES_STATE'
```

AppLogic code snippet:

```
IDataConnSet connSet;
connSet = createDataConnSet(0);

// Create database connections
IDataConn conn_detailDB = createDataConn(0,
GX_DA_DAD_DRIVERS.GX_DA_DRIVER_DEFAULT, "kdetails", "", "kuser",
"kpassword");

IDataConn conn_statesDB = createDataConn(0,
GX_DA_DAD_DRIVERS.GX_DA_DRIVER_ODBC, "kstates", "", "kuser",
"kpassword");

// Specify query / db connection pairs
```

```
connSet.addConn("DETAILS", conn_detailDB);
connSet.addConn("STATES", conn_statesDB);

// Create IValList that contains the REGION parameter
// value to pass to the hierarchical query
IValList param = GX.CreateValList();
param.setValString("REGION", "WEST");

IHierQuery hqry;
// Load the GXQ file with the db connection set and
// parameter value
hqry = loadHierQuery("state.gxq", connSet, 0, param);

// Run the report
evalTemplate("state.html", hqry);
```

Related Topics

createDataConnSet( ),
IDataConnSet interface,
IHierQuery interface

loadQuery( )

Creates a flat query by loading a query file.

loadQuery( ) is deprecated. Instead, use loadQuery( ) from the class com.kivasoft.util.SqlUtil.

Syntax

```
public IQuery loadQuery(
    String pFileName,
    String pQueryName,
    int flags,
    IValList pParams)
```

pFileName. Name of the query (.GXQ) file, including the path. Use a relative path when possible.

A query file is an ASCII text file containing one or more SQL statements. You can create the file using any ASCII text editor. Use the following syntactical guidelines:

- The query file for a flat query contains a SQL SELECT statement (compliant with ANSI SQL89) preceded by the following line:

```
/* optional comments */
```

```
query queryName using (driverCode, DSN, UserName) is
```

where *queryName* is the name of the flat query. Do not use any semicolons (;) in the query file.

- In the query file, do not use any semicolons (;) or other vendor-specific SQL statement terminators. The SQL statement may contain placeholders in the WHERE clause.

pQueryName. Name of the query in the query file.

flags. Specify 0 (zero). Internal use only.

pParams. IValList of query file parameters, or null. A collection of placeholders for the WHERE clause. A placeholder may be a name or a number. It is prefixed by a colon (:) character. The placeholders can be replaced by specifying replacement values in the IValList parameter.

Usage

Use loadQuery( ) to create a flat query object by loading a query (.GXQ) file. An AppLogic can retrieve standardized queries stored in a data file and, at runtime, can dynamically select and assign the data source on which the query is run.

You create the query file separately using the Query Designer or an ASCII text editor, ANSI 89 standard SQL SELECT statements, and special syntax.

To run the flat query, call executeQuery( ) in the IDataConn Interface (deprecated).

Return Value

IQuery object, or null for failure (such as query file not found).

Example

The following example shows a query (GXQ) file and a section of an AppLogic that loads and executes the query:

Query file:

```
/* STATES */
query STATES using (ODBC, kstates, kuser) is
select STATES.STATE as STATES_STATE
from STATES
where (STATES.REGION = ':REGION')
order by STATES.STATE asc
```

AppLogic code snippet:

```
// Create database connection
IDataConn conn = createDataConn(0,
GX_DA_DAD_DRIVERS.GX_DA_DRIVER_DEFAULT, "kstates", "", "kuser",
"kpassword");

// Create IValList that contains the REGION parameter
// value to pass to the query
IValList param = GX.CreateValList();
param.setValString("REGION", "WEST");

IQuery qry;
// Load the query file with the parameter value
qry = loadQuery("state.gxq", "STATES", 0, param);

// Execute the query
IResultSet rs = conn.executeQuery(
GX_DA_EXECUTEQUERY_FLAGS.GX_DA_RS_BUFFERING, qry, null, null);
```

Related Topics

IQuery Interface (deprecated)

log( )

Writes a message to the server log.

log( ) is deprecated. In standard-based applications, use one of the log( ) methods available from the javax.servlet package. For example, refer to the GenericServlet class or the ServletContext interface.

Syntax 1

Logs a message (type = GXLOG.GXEVENTTYPE_INFORMATION and category = 0).

```
public int log(
   String msg)
```

Syntax 2

Logs an event with a message, specifying the type and category of event.

```
public int log(
   int type,
   int category,
   String msg)
```

msg. Message text to log.

type. Message type. Use one of the following variables:

- GXLOG.GXEVENTTYPE_INFORMATION
- GXLOG.GXEVENTTYPE_ERROR
- GXLOG.GXEVENTTYPE_SYSTEM
- GXLOG.GXEVENTTYPE_WARNING

category. User-defined message category. Do not use the range of values reserved for the iPlanet Application Server systems, which is 0 to 65535, inclusive.

Usage

Use log( ) for displaying or storing simple messages or for debugging. The output can be directed to the console, to a text file, or to a database table. To direct output, use the iPlanet Application Server Administrator. For more information, see the *Administration and Deployment Guide*.

Return Value

GXE.SUCCESS if the method succeeds.

```
Example 1
// Log a message
log("This is the message text saved to the log");
Example 2
// Log messages that include String variables
log(firstName+lastName+password+" "+String.valueOf(cusId));
log(GXLOG.GXEVENTTYPE_ERROR", -1, Cannot find table:
"+"Shipping");
```

loginSession( )

Logs an authorized user into a session with a secured application.

loginSession( ) is deprecated. As of the 2.1 servlet specification, there is no standard security model for servlets. However, iPlanet Application Server provides a security model. In servlets, use the equivalent loginSession( ) method, available from the HttpSession2 Interface in package com.iplanet.server.servlet.extension.

Syntax 1

Use in most cases.

```
public int loginSession(
   String name,
   String password)
```

Syntax 2

Contains several parameters that are placeholders for future functionality.

```
public int loginSession(
   String pDomain,
   int dwMethod,
   int dwFlags,
   String pName,
   byte[] pAuthData,
   int nAuthData)
```

name. The login user name.

password. The user password.

pDomain. Specify null.

dwMethod. Specify 0.

dwFlags. Specify 0.

pName. The login user name.

pAuthData. The user password.

nAuthData. The size of the password.

Usage

Call loginSession( ) after creating a user session with createSession( ) or after retrieving a user session with getSession( ). loginSession( ) checks the passed in login name and password against the user names and passwords stored in the iPlanet Application Server (the administrator sets up and manages this information) and logs the user into the session if the login name and password are valid.

If login is successful, a security credential object is created and associated with the session. The server checks this security credential object each time it receives an AppLogic request, and verifies if the user has execute permission for the AppLogic.

Using loginSession( ) in conjunction with isAuthorized( ), an application can ensure that only authorized users can execute certain AppLogics or take certain actions.

Tip

The server administrator creates users and passwords and manages access to AppLogics and specified resources, such as sales or forecast reports. During the development and debugging phases, application developers can use the ldapmodify tool to create users, groups, and ACLs in the LDIF file. These tasks cannot be done programmatically.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
public class Welcome extends AppLogic
{
    public int execute()
   {
      try {

         // Check user login
         if (valIn.getValString("NAME") == null ||
            valIn.getValString("PASSWORD") == null) {
            log("missing login NAME/PASSWORD");
            return evalOutput("kivaapp/shop/login_again",
                  (ITemplateData) null,
                  (ITemplateMap)  null, null, null);
         }
         // If login succeeds, create a session
         ISession2 mySess;
         mySess = getSession(0, null, null);
         if (mySess == null) {
            mySess=createSession(0, 60000, null, null, null);
            log("created session: " + String.valueOf(mySess != null));
         } else
            log("got session");

         // Now, look up user NAME/PASSWORD in database
         // and see what role the user has.  The database
         // should have a user table which tracks all the
         // users of the online shop application.
         //
         String role;
         role = /* Database lookup here. */ "Shop_Customer";

         // Call loginSession() to set up the session with
         // that role. Future requests to AppLogics in this
         // session will now operate under the right role.
         //
         loginSession(role, "");
         saveSession(null);

         // Check to see if the current role is authorized
         // against some of the more advanced operations,
```

```
         // and choose the appropriate main menu page to
         // return to the user.
         //
         if (isAuthorized("Shop_Inventory", "READ") ==
            GXACLPERMSTATUS.GXACL_ALLOWED ||
            isAuthorized("Shop_Daily_Forecast", "READ") ==
            GXACLPERMSTATUS.GXACL_ALLOWED ||
            isAuthorized("Shop_Weekly_Forecast", "READ") ==
            GXACLPERMSTATUS.GXACL_ALLOWED)
            return evalOutput("mainmenu_advanced",
                  (ITemplateData) null,
                  (ITemplateMap)  null, null, null);
         return evalOutput("kivaapp/shop/mainmenu_regular",
               (ITemplateData) null,
               (ITemplateMap)  null, null, null);
      } finally {
      }
   }
}
```

Related Topics

isAuthorized( ),
logoutSession( )

logoutSession( )

Logs a user out of a session with a secured application.

logoutSession( ) is deprecated. As of the 2.1 servlet specification, there is no standard security model for servlets. However, iPlanet Application Server provides a security model. In servlets, use the equivalent isAuthorized( ) method, available from the HttpSession2 Interface in package com.iplanet.server.servlet.extension.

Syntax

```
public int logoutSession(
   int dwFlags)
```

dwFlags. Specify 0.

Usage

If the AppLogic called loginSession( ) to log into a session with a secured application, call logoutSession( )when the user exits the application, or the secured portion of it.

Rule

Call getSession( ) before calling logoutSession( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

getSession( ),
isAuthorized( ),
loginSession( )

newRequest( )

Calls another AppLogic from within the current AppLogic.

Syntax 1

Passes in the ValIn and ValOut of the current AppLogic to the called AppLogic. If the called AppLogic streams results, the calling AppLogic streams the same results.

```
public int newRequest(
   String guidSTR)
```

Syntax 2

Passes in the specified valIn and valOut.

```
public int newRequest(
   String guidSTR,
   IValList vIn,
   IValList vOut)
```

Syntax 3

Use to explicitly specify the location of AppLogic execution.

```
public int newRequest(
   String guidSTR,
   IObject vIn,
   IObject vOut,
   int host,
   int port,
   int flag)
```

guidSTR . String GUID or name of the AppLogic to execute.

vIn. IValList object containing input parameters to pass to the called AppLogic.

vOut. IValList object containing result values of the called AppLogic.

host. IP address of the Internet host of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

port. Internet port of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

flag. Specify zero.

Usage

Use newRequest( ) to call another AppLogic from within the current AppLogic. When it calls newRequest( ), the AppLogic passes to the iPlanet Application Server the GUID or name of the AppLogic to execute and, optionally, any input and output parameters.

iPlanet Application Server constructs a request using the parameters specified and processes it like any other request, by instantiating the AppLogic and passing in its parameters. The results from the called AppLogic module are returned to the calling AppLogic.

The AppLogic that newRequest( ) invokes can do one of the following tasks:

- Process application logic and return result values in the vOut parameter.
- Process application logic and return the resulting data form (such as a report) by streaming the output or by calling result( ).
- Process application logic and return result values in the vOut parameter as well as return the resulting data form (such as a report) by streaming the output or by calling result( ).

If the called AppLogic uses evalOutput( ) to stream results, evalOutput( ) returns HTML results by default. The current AppLogic can, however, specify that evalOutput( ) return a non-HTML data stream by setting the gx_client_type key to "ocl" in the input IValList of newRequest( ). For example:

```
vallist.setValString("gx_client_type", "ocl");
```

Rule

The specified GUID string, input parameters, and output parameters must be valid for the specified AppLogic.

Tips

- The calling AppLogic can create new input and output IValLists so as to avoid changing its own input and output IValLists.

- The AppLogic can call another AppLogic, passing its own input and output IValLists. In this case, the called AppLogic accesses the same stream destinations as the calling AppLogic.
- Use newRequestAsync( ) instead of newRequest( ) to execute asynchronous request.
- Called AppLogics might reside on different servers, depending on partitioning and load balancing configurations, might be written in a different language, or might have cached results. The calling AppLogic can be unaware or independent of these conditions.
- Using newRequest( ), you can modularize parts of the application, build dynamic header/footer information and smart reporting templates, and hide complex or confidential business logic in secure submodules or even separate servers.
- Use newRequest( ) judiciously. Each invoked AppLogic uses a certain amount of communications and server resources.

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Call specified AppLogic and pass parameters
newRequest("{E5CA1000-6EEE-11cf-96FD-0020AFED9A65}",
   paramsToModule, paramsReturned);
```

Example 2

```
// Use DisplayBasket AppLogic to display the contents
if(newRequest("DisplayBasket", valIn, valOut)==0){
   return 0;
   }
else
   return result("Cannot execute DisplayBasket AppLogic");
```

Related Topics

GUID class,
IValList interface

newRequestAsync( )

Calls another AppLogic from within the current AppLogic, and runs it asynchronously.

Syntax 1

Passes in the ValIn and ValOut of the AppLogic.

```
public IOrder newRequestAsync(
   String guidSTR)
```

Syntax 2

Passes in the specified valIn and valOut.

```
public IOrder newRequestAsync(
   String guidSTR,
   IValList vIn,
   IValList vOut)
```

Syntax 3

Use to explicitly specify the location of AppLogic execution.

```
public IOrder newRequest(
   String guidSTR,
   IObject vIn,
   IObject vOut,
   int host,
   int port,
   int flag)
```

guidSTR . String GUID or name of the AppLogic to execute.

vIn. IValList object containing input parameters to pass to the called AppLogic.

vOut. IValList evalOutput( ) object containing result values of the called AppLogic.

host. IP address of the Internet host of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

port. Internet port of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

flag. Specify 0.

Usage

Use newRequestAsync( ) to call another AppLogic from within the current AppLogic, and run it asynchronously. Executing an AppLogic asynchronously is useful if the AppLogic performs a lengthy operation, or if the AppLogic acts as a monitor or remains persistent. For example, an asynchronous AppLogic may perform a lengthy database query to produce a complex result set that it sends an e-mail to a destination address. Another AppLogic module may run continuously and re-index HTML pages every 24 hours.

When an AppLogic calls newRequestAsync( ), it passes to the iPlanet Application Server the GUID of the AppLogic module to execute and, optionally, any input and output parameters.

The iPlanet Application Server constructs a request using the parameters specified and processes it like any other request, by instantiating the AppLogic and passing in its parameters. The results from the called AppLogic module are returned to the calling AppLogic.

The AppLogic that newRequestAsync( ) invokes can do one of the following tasks:

- Process application logic and return result values in the vOut parameter.
- Process application logic and return the resulting data form (such as a report) by streaming the output or by calling result( ).
- Process application logic and return result values in the vOut parameter as well as return the resulting data form (such as a report) by streaming the output or by calling result( ).

Rules

- The specified AppLogic must be accessible to the iPlanet Application Server.
- The specified GUID string, input parameters, and output parameters must be valid for the specified AppLogic module.

Tips

- To get the current status of the request, use the getState( ) method in the returned IOrder object.
- The calling AppLogic can use GX.WaitForOrder( ) to wait for one or multiple asynchronous requests to return.
- The calling AppLogic can create new input and output IValLists so as to avoid changing its own input and output IValLists.

- The AppLogic can call another AppLogic, passing its own input and output IValLists. In this case, the called AppLogic accesses the same stream destinations as the calling AppLogic. To prevent conflicts in streaming, the calling AppLogic can use GX.WaitForOrder( ) to wait until the called AppLogic is finished.
- Using newRequestAsync( ), you can modularize parts of the application, build dynamic header / footer information and smart reporting templates, and hide complex or confidential business logic in secure submodules or even separate servers.
- Use newRequestAsync( ) judiciously. Each invoked AppLogic uses a certain amount of communications and server resources.

Return Value

IOrder object, or null for failure.

Example

```
Orders[] = new IOrder[1];
int nOrder;

Orders[0] = newRequestAsync(asyncGUIDStr, valIn, valOut);
if (Orders[0] != null)
{
   log("Successfully invoked async AppLogic\n");

   // wait for async applogic to finish (max 100 seconds)
   nOrder = GX.WaitForOrder(Orders, context, 100);
   if (nOrder >= 0)
   {
       return result("Error in executing async request:
         order wait returned an error");
   }
   else
   {
      getStateIOrder state = Orders[0].getState();
      if (state == null || state.pdwState != GXE.SUCCESS)
         return result("Error in executing async
            request");
   }
}
else
{
   log("Failed to invoke async AppLogic\n");
}
```

Related Topics

GUID class,
IOrder interface,
IValList interface

removeAllCachedResults( )

Clears an AppLogic's result cache.

Syntax

```
public int removeAllCachedResults(
   String guid)
```

guid. The guid that identifies the AppLogic whose result cache to clear. Specify null to clear the current AppLogic's cache.

Usage

To free system resources, use removeAllCachedResults( ) to clear an AppLogic's result cache when the results are no longer needed. This method clears the cache, but does not disable caching.

Tips

- To clear an AppLogic's entire result cache and discontinue caching, use deleteCache( ).
- To clear a specific result from the cache, use removeCachedResult( ).

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
int hr;
String guid = valIn.getValString("applogic");

hr = removeAllCachedResults(guid);
if (hr == GXE.SUCCESS)
   return result("Successfully cleared cached results")
else
   return result("Failed to clear cached results");
```

Related Topics

deleteCache( ),
removeCachedResult( ),
setCacheCriteria( )

removeCachedResult( )

Clears a specific result from an AppLogic's result cache.

Syntax

```
public int removeCachedResult(
    String guid,
    IValList criteria)
```

guid. The guid that identifies the AppLogic whose cached result to clear. Specify null to clear the current AppLogic's cached result.

criteria. An IValList object that contains the criteria for selecting the result to remove. In the IValList object, set a specific value that matches the cache criteria passed to setCacheCriteria( ). For example, if the cache criteria passed to setCacheCriteria( ) was "Salary=40000-60000"), you can remove results where salary is 50000 by setting in the IValList object a "Salary" key to a value of "50000".

Usage

Use removeCachedResult( ) to clear a specific result from an AppLogic's cache when the result is no longer needed.

Tips

- To clear an AppLogic's entire result cache and discontinue caching, use deleteCache( ).
- To clear an AppLogic's entire result cache, but continue caching, use removeAllCachedResults( ).

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
int hr;
String guid = valIn.getValString("applogic");

IValList resultList = GX.CreateValList();
resultList.setValString("Salary", "50000");
```

```
hr = removeCachedResult(guid, resultList);

if (hr == GXE.SUCCESS)
   return result("Successfully deleted specified result")
else
   return result("Failed to delete specified result");
```

Related Topics

deleteCache( ),
removeAllCachedResults( ),
setCacheCriteria( )

result( )

Specifies the return value of an AppLogic.

result( ) is deprecated. In standard-based applications, use similar functionality as defined in the Servlet API. For example, refer to javax.servlet.ServletOutputStream or javax.servlet.http.HttpServletResponse.

Syntax

```
public int result(
   String result)
```

result. Text representing the result value of the current AppLogic.

Usage

Use result( ) in conjunction with the execute( ) method to define a return value for an AppLogic. In general, use result( )in an AppLogic that services HTTP or HTML requests and returns a simple HTML string that does not require streaming.

In the execute( ) method, the AppLogic can call result( )in conjunction with the return statement to send data results directly back to the entity that called the AppLogic.

Rule

An AppLogic can stream results using streamResultHeader( ) or streamResult( ) . If the AppLogic streams results, call result( ) only *after* finishing streaming.

Tips

- To construct HTML output programmatically, use the Java StringBuffer class to efficiently build up an HTML result, which can then be efficiently converted to a string to be passed in as the input to result( ).
- An AppLogic can cache results for reuse using setCacheCriteria( ).
- Alternatively, the AppLogic can return results using a template. The AppLogic can call evalOutput( ) to merge a dynamically created result set from a hierarchical query with a template to produce formatted results. The result from evalOutput( ) is streamed automatically.

Return Value

GXE.SUCCESS if the method succeeds.

```
Example 1
// Return a simple HTML string
public class HelloWorld extends AppLogic {
   public int execute() {
      return result("Hello, world!\n");
   }
}
Example 2
// Dynamically create the return string
public int execute() {
   String resultStr;
   String text;
   text = valIn.getValString("Incoming Text");
   if (text == null)
      resultStr = "No input text.";
   else if (text.indexOf('-') >= 0)
      resultStr = "Input text has a hyphen.";
   else
      resultStr = "Input text has no hyphen.";
   resultStr = "<HTML>" + resultStr + "</HTML>";
   return result(resultStr);
}
```

Related Topics

execute( ),
streamResult( ),
streamResultHeader( )

saveSession( )

Saves changes to a session.

saveSession( ) is deprecated. In standard-based applications, this method is unnecessary because the functionality is automatically provided.

Syntax

```
public int saveSession(
    ISessionIDGen pIDGen)
```

pIDGen. The session ID generation object used to generate session IDs. Specify null.

Usage

Use saveSession( ) to ensure that changes are saved in the distributed state storage area, which stores the session information for subsequent use if any other AppLogics are invoked within the same session.

The saveSession( ) method uses a cookie—if the Web browser supports cookies—to pass the session ID back and forth between the Web browser and iPlanet Application Server. It transfers only the session ID, not the session information itself, to provide better information security.

Because saveSession( ) uses streamResultHeader( ) to register the cookie, be sure to call saveSession( ) before calling streamResult( ), evalTemplate( ), or any other HTTP body streaming methods.

Tip

- The AppLogic needs to call the saveSession( ) method in the AppLogic class at least once to set a cookie. The saveSession( ) method in the ISession2 interface only saves data to the distributed state store, whereas saveSession( ) in the AppLogic class saves data to the distributed state store *and* sets a cookie.
- The AppLogic should call saveSession( ) to save changes after updating session data.
- To improve performance, keep smaller amounts of information in the session.

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics
createSession( ),
getSession( ),
Session2 class ,
ISession2 interface

setCacheCriteria( )

Stores AppLogic results, such as HTML, data values, and streamed data, in a result cache.

setCacheCriteria( ) is deprecated. In iPlanet Application Server 6.0 applications, this functionality is controlled using the setCacheCriteria property in the servletInfo.ntv file.

Syntax

```
public int setCacheCriteria(
    int timeout,
    int cachesize,
    String criteria)
```

timeout. Number of seconds the AppLogic result remains in the result cache after the last access. To clear the result cache after a specified time from its creation, use the GXREPOSIT.GXREPOSIT_TIMEOUT_CREATE flag, as shown in the following example: `setCacheCriteria(GXREPOSIT.GXREPOSIT_TIMEOUT_CREATE | 300, . . . )`. In this example, the cache is cleared 300 seconds after it is created. Set timeout to zero to clear the result cache and disable caching for this AppLogic.

cachesize. Maximum number of results to be cached for the AppLogic at any time. The result cache stores distinct AppLogic output up to the cachesize limit. If the AppLogic generates another output to cache, the least accessed member of the cache is dropped. Setting cachesize to zero clears the result cache and disables caching for this AppLogic.

criteria. Criteria expression containing a string of comma-delimited descriptors. Each descriptor defines a match with one of the input parameters to the AppLogic. Use the following syntax:

Syntax	Description
`arg`	Test succeeds for any value of *arg* in the input parameter list. For example: `setCacheCriteria(3600,1,"EmployeeCode");`

Syntax	Description

`arg=v`	Test whether *arg* matches *v* (a string or numeric expression). For example:
	`"stock=NSCP"`
	Assign an asterisk (*) to the argument to cache a new set of results every time the AppLogic module runs with a different value. For example:
	`setCacheCriteria(3600,1,"EmployeeCode=*");`

`arg=v1|v2`	Test whether *arg* matches any values in the list (*v1*, *v2*, and so on). For example: `"dept=sales|marketing|support"`
`arg=n1-n2`	Test whether *arg* is a number that falls within the range. For example: `"salary=40000-60000"`

Usage

Use to specify caching for the results from an AppLogic. An AppLogic can cache any type of result. Caching improves performance for time-consuming operations such as queries and report generation.

When caching is enabled for an AppLogic, the iPlanet Application Server stores its input parameter values and its results in the cache so that, if the AppLogic is called again with the same parameters (matching the cache criteria), the iPlanet Application Server retrieves its results directly from the cache instead of running the AppLogic again. If the AppLogic is called with different parameters, the iPlanet Application Server runs the AppLogic again and saves its result in the cache as well.

Each AppLogic has only one cache but it can contain multiple sets of results if the AppLogic was run multiple times with different parameters for each call.

Tips

- Do not use caching if real-time results are needed. For example, to ensure current data, caching is not recommended for query operations on highly volatile data.
- Use skipCache( ) to bypass result caching if an error occurred during AppLogic execution.
- Use isCached( ) to test whether caching is currently enabled. Calling isCached( ) is important because it prevents calling setCacheCriteria( ) too many times.

- To change the caching criteria for AppLogic, call setCacheCriteria( ) again, this time specifying different caching criteria. Each subsequent call supersedes the previous call, discarding the current contents of the result cache, and its criteria remain in effect until the next setCacheCriteria( ) call, if applicable.
- To stop caching results, call deleteCache( ). A subsequent call to setCacheCriteria( ) can reactivate caching.

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Verify AppLogic caching before setting cache criteria
if(!isCached())  {
   log("Set criteria to save output from 3 deptcodes");
   if(setCacheCriteria(60, 3, "deptcode")!=0)
      log("Could not set criteria");
   else
      log("Succeeded in setting criteria");
}
else
   log("Not Setting Criteria");
```

Example 2

```
// Cache multiple results for up to 100 values of Department

setCacheCriteria(3600,100,"Department");
```

Example 3

```
// Cache single result for given matching value of Department

setCacheCriteria(3600,1,"Department=Operations");
```

Example 4

```
// Cache multiple results for two matching values of dept

setCacheCriteria(3600,2,"Department=Research | Engineering");
```

Example 5

```
// Cache one result for salary in a range

setCacheCriteria(3600,1,"Salary=40000-60000");
```

Example 6

```
// Cache two results for several parameters
```

```
setCacheCriteria(3600, 2,
    "Department=Sales,Salary=40000-60000");
```

Related Topics

deleteCache( ),
removeAllCachedResults( ),
removeCachedResult( ),
isCached( ),
skipCache( )

setSessionVisibility( )

Sets the session visibility.

setSessionVisibility( ) is deprecated. It provides functionality that does not apply to standard-based applications.

Syntax

```
public int setSessionVisibility(
   String domain,
   String path,
   boolean isSecure)
```

domain. The domain in which the session is visible.

path. The path to which this session must be visible.

isSecure. If TRUE, the session is visible only to secure servers (HTTPS).

Usage

Because of the way cookies are used to identify sessions, iPlanet Application Server sessions are, by default, accessible only within the same URL name space where they were created. As a result, if you call only the saveSession( ) method, then your session is not visible to any other domain or URL.

However, if you call setSessionVisibility( ) before calling saveSession( ), you can control the visibility of the session. The setSessionVisibility( ) method internally controls the attributes of the cookie used in transmitting the session ID.

You must be part of the domain to set the domain attribute. For example, if the domain is set to iplanet.com, then the session is visible to foo.iplanet.com, bar.iplanet.com, and so on. Domains must have at least two periods (.) in them. For example, .net is an invalid domain attribute.

By default, the session is visible only to the URL that created the session cookie. Use the path parameter to specify different URLs that will be visible. For example, the path /phoenix would match "/phoenixbird" and "/phoenix/bird.html". To make the entire server root visible, specify a path of "/", the most general value possible.

Both the domain and path parameters are null-terminated character strings. They are not modified within the setSessionVisibility( ) method.

Rule

For the session visibility to take effect, you must invoke setSessionVisibility( ) before a call to saveSession( ). The saveSession( )method uses the visibility attributes set from setSessionVisibility( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

saveSession( )

setVariable( )

Sets a value that is passed to later AppLogic requests that are called by the same client. If the client is a browser, cookies are used to transfer variable values.

setVariable( ) is deprecated. In standard-based applications, use the addCookie( ) method in the interface javax.servlet.http.HttpServletResponse.

Syntax 1

```
public int setVariable(
   String name,
   String value)
```

Syntax 2

```
public int setVariable(
   String name,
   String value,
   int timeout,
   String urlPath,
   String urlDomain,
   boolean secure)
```

name. The name of the value to record for this browser session. The value will appear on any future AppLogic's input IValList under this name.

value. The string value to record.

timeout. Number of seconds before the cookie expires. Applies to HTTP clients only.

urlPath. The subset of URLs in a domain for which the cookie is valid. Applies to HTTP clients only.

urlDomain. The domain for which the cookie is valid. Applies to HTTP clients only.

secure. If a cookie is marked secure, it will be sent only if the communications channel with the host is a secure one. Currently, this means that secure cookies will be sent only to HTTPS (HTTP over SSL) servers. Applies to HTTP clients only.

Usage

Use setVariable( ) to store information specific to a client that you want to pass to other AppLogics invoked by the same client. The values set with setVariable( ) are passed to the input IValList (valIn) of the called AppLogics.

In the case of an HTTP client, setVariable( ) streams the variable out in an HTTP header. The HTTP header registers a cookie, which is the mechanism used to pass data back and forth between the browser and the iPlanet Application Server.

Rule

Because setVariable( ) streams information in an HTTP header, call itbefore calling any HTTP body streaming methods, such as streamResult( ), evalOutput( ), and evalTemplate( ).

Tip

If your application requires more security, you should use iPlanet Application Server's session mechanism instead of cookies to maintain session information. With a iPlanet Application Server session, data is stored on the server and only a session ID is passed between the client and the server. For more information about the session mechanism, see ISession2 interface.

Return Value

GXE.SUCCESS if the method succeeds.

skipCache( )

Skips result caching for the current AppLogic execution.

Syntax

```
public int skipCache()
```

Usage

Use skipCache( ) to prevent results from the current request from being saved in the results cache if an error occurs during AppLogic execution.

Rule

For skipCache( ) to have any effect, you must first enable caching by calling setCacheCriteria( ).

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Skip result caching if an error occurs
// during AppLogic execution
public class produceReport extends AppLogic {
   public int execute() {
   if (isCached == false)
      // Set up results cache
      setCacheCriteria(3600, 1, "");
   String result;
   if (. . . Input arguments are not valid . . .) {
      result = "Please enter your employee id...";

      // We don't want the above error prompt to be
      // put into the result cache, so we must
      // call skipCache.
      skipCache();
      else {
      result = [. . . Generate report . . .];
   }
   return result(result);
   }
}
```

Related Topics

deleteCache( ),
removeAllCachedResults( ),
removeCachedResult( ),
isCached( ),
setCacheCriteria( ),
IValList interface

streamResult( )

Streams results as a string.

streamResult( ) is deprecated. In standard-based applications, use similar functionality as defined in the Servlet API. For example, refer to javax.servlet.ServletOutputStream or javax.servlet.http.HttpServletResponse.

Syntax

```
public synchronized int streamResult(
   String res)
```

res. The body data to stream. If returning HTML body data, you can use HTML formatting following HTTP body conventions. See your HTTP documentation for more information.

Usage

Use streamResult( ) to stream data as soon as it is available. With streaming, an AppLogic can make the first portion of the data available for use immediately, even if the remainder of the stream has not yet been processed. This is especially useful with large volumes of data, such as a query that takes a while for the database server to process completely. An AppLogic can process and display those rows in the result set that have been returned. Without streaming, AppLogic must prepare the entire result first before returning any data.

The streamResult( ) method is typically used to stream HTTP body content. Before calling streamResult( ), the AppLogic must call streamResultHeader( ) to return the HTTP header data first. The HTTP protocol separates data streams into header and body data, and specifies that the header data and body data are returned in that order. For details about HTTP header and body data, see your HTTP documentation.

Tips

- Alternatively, use evalTemplate( ) to stream HTTP body output. It merges data with an HTML template. As soon as a segment of the output page is finished, evalTemplate( ) streams it out to the Web browser.
- An AppLogic can call streamResultHeader( ) and streamResult( ) repeatedly to stream more results.
- To stream binary data, use streamResultBinary( ).

Return Value

GXE.SUCCESS if the method succeeds.

```
Example 1
// Stream header and body, passing the header and body data
// as variables.
streamResultHeader(headerVariable);
streamResult(bodyVariable);
Example 2
// Stream header or body in several parts, using several
// method calls, starting with the header
streamResultHeader(startHeader);
streamResultHeader(finishHeader);
streamResult(bodyStart);
streamResult(bodyMiddle);
streamResult(bodyEnd);
```

Related Topics

streamResultBinary( ),
streamResultHeader( )

streamResultBinary( )

Streams binary data, such as a GIF file.

streamResultBinary( ) is deprecated. In standard-based applications, use similar functionality as defined in the Servlet API. For example, refer to javax.servlet.ServletOutputStream or javax.servlet.http.HttpServletResponse.

Syntax

```
public synchronized int streamResultBinary(
   byte[] buf,
   int offset,
   int length)
```

buf. The array from which binary data is streamed.

offset. Index in the array. The starting position in the array to start streaming binary body data.

length. Number of bytes to stream from the array, starting at the specified offset position.

Usage

Use streamResultBinary( ) to stream binary data as soon as it is available. With streaming, an AppLogic can make the first portion of the data available for use immediately, even if the remainder of the stream has not yet been processed. This is especially useful with large volumes of data, such as a query that takes a while for the database server to process completely. An AppLogic can process and display those rows in the result set that have been returned. Without streaming, AppLogic must prepare the entire result first before returning any data.

The streamResultBinary( ) method is used to stream HTTP body data of binary type, such as an image (GIF) file. Before calling streamResultBinary( ), the AppLogic should call streamResultHeader( ) to return the HTTP header data first. The HTTP protocol separates data streams into header and body data, and specifies that the header data and body data are returned in that order. For details about HTTP header and body data, see your HTTP documentation.

Tips

- Alternatively, use evalTemplate( ) to stream HTTP body output. It merges data with an HTML template. As soon as a segment of the output page is finished, evalTemplate( ) streams it out to the waiting Web browser.
- To stream non-binary data, use streamResult( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

streamResult( ),
streamResultHeader( )

streamResultHeader( )

Streams header data.

streamResultHeader( ) is deprecated. In standard-based applications, use similar functionality as defined in the Servlet API. For example, refer to javax.servlet.ServletOutputStream or javax.servlet.http.HttpServletResponse.

Syntax

```
public synchronized int streamResultHeader(
   String hdr)
```

hdr. The header data to stream. If returning HTTP header data, use the HTTP header conventions, such as the following:

```
"Content-Type: text/html"

"Location: <redirect url>"
```

See your HTTP documentation for more information.

Usage

Use streamResultHeader( ) to return header data before streaming body data. With streaming, an AppLogic can make the first portion of the data available for use immediately, even if the remainder of the stream has not yet been processed. This is especially useful with large volumes of data, such as a query that takes a while for the database server to process completely. An AppLogic can process and display those rows in the result set that have been returned. Without streaming, AppLogic must prepare the entire result first before returning any data.

The streamResultHeader( ) method is typically used in conjunction with streamResult( ) to stream HTTP data. Before calling streamResult( ), the AppLogic should call streamResultHeader( ) to return the HTTP header data first. The HTTP protocol separates data streams into header and body data, and specifies that the header data and body data are returned in that order. For details about HTTP header and body data, see your HTTP documentation.

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Stream header and body, passing the header and body data
// as variables.
StreamResultHeader(headerVariable);
StreamResult(bodyVariable);
```

Example 2

```
// Stream header or body in several parts, using several
// method calls, starting with the header
StreamResultHeader(startHeader);
StreamResultHeader(finishHeader);
StreamResult(bodyStart);
StreamResult(bodyMiddle);
StreamResult(bodyEnd);
```

Related Topics

streamResult( ),
streamResultBinary( )

BaseUtils class

The BaseUtils class provides a set of convenience methods similar to those in the GX class. The methods of this class apply to user-written AppLogics.

Although anyone developing a iPlanet Application Server application can use BaseUtils, this class is typically used in components generated by iPlanet Application Builder.

Package

com.iplanet.server.servlet.extension

Constructor

public BaseUtils( )

Methods

Method	Description
convertITemplateDataToResultSet( )	Creates a ResultSet wrapper for an ITemplateData object.
createListRowSet( )	Creates a RowSet object that conforms to the IListRowSet interface.
createMethodHash( )	Creates a method hash.
createStringFromBuffer( )	Creates a string from an IBuffer object.
doubleQuote( )	Converts each occurrence of one single quote to two successive single quotes.
includeJSP( )	A convenience method for streaming a JSP file.
initRowSets( )	Initializes RowSet objects specified in a servlet's configuration file.
log( )	Calls the log( ) method in the AppLogic class and logs an error to the error output.

convertITemplateDataToResultSet( )

Creates a ResultSet wrapper for an ITemplateData object.

Syntax

```
public static ResultSet convertITemplateDataToResultSet(
    String groupName,
    ITemplateData templateData)
```

groupName. The group name of the specified ITemplateData.

templateData. The ITemplateData to be wrapped.

Usage

The convertITemplateDataToResultSet( ) method takes an ITemplateData object and wraps it so that it conforms to the specifications of a JDBC ResultSet object.

Rules

Hierarchical ITemplateData objects are not supported, so do not specify one as a parameter.

Return Value

A ResultSet object.

createListRowSet( )

Creates a RowSet object that conforms to the IListRowSet interface.

Syntax

```
public static IListRowSet createListRowSet(
    HttpServletRequest request,
    HttpServletResponse response,
    String listName)
```

request. The request object.

response. The response object.

listName. The name of the RowSet to pass in.

Usage

The createListRowSet( ) method creates a RowSet object that conforms to the IListRowSet interface. IListRowSet objects are useful for dynamically populating a pop-up list. The listName parameter is used to add the RowSet name as an attribute of the specified request.

Return Value

An IListRowSet object.

Related Topics
IListRowSet interface

createMethodHash( )
Creates a method hash.

Syntax
```
public static final MethodHash createMethodHash()
```

Usage
Method hashes are used to cache method lookups but are hidden from the user. createMethodHash( ) allows for the construction of method hashes. This construction is necessary when subclasses of DBRowSet are created.

Return Value
A method hash.

Related Topics
getStaticMethodCache( ) in the DBRowSet class

createStringFromBuffer( )
Creates a string from an IBuffer object.

Syntax
```
public static final String createStringFromBuffer(
   IBuffer buff)
```

buff. IBuffer object containing the value to convert.

Usage
This method creates a String from an IBuffer object. Use createStringFromBuffer( ) instead of the toString( ) method of the Util class. Unlike toString( ), the createStringFromBuffer( ) method accounts for null terminators. Ignoring null terminators in an IBuffer object causes incorrect Strings.

Tip
createStringFromBuffer( ) does not perform character conversion. You may want to write your own method if character conversion is important.

Return Value
String representing the data in an IBuffer object.

Example

```
ITemplateMap map = (ITemplateMap) request.getAttribute("GX_MAP");

IBuffer buff;

buff = map.get("foo",null,null);
```

Related Topics

Util class,
IBuffer interface

doubleQuote( )

Converts each occurrence of one single quote to two successive single quotes.

Syntax

```
public static final String doubleQuote(
   String input)
```

input. The input string whose single quotes are to be affected.

Usage

Use doubleQuote( ) for input bindings to queries. Suppose there is a query that takes one input parameter, NameInput:

```
select *

from emp

where (ename = ':NameInput')
```

If the name is O'Reilly, then the following query would be loaded:

```
select *

from emp

where (ename ='O'Reilly')
```

Normally, an attempt to execute this query results in a syntax error. The correct query should be as follows:

```
select *

from emp

where (ename ='O''Reilly')
```

To produce this query, any single quotes in the input binding for NameInput must be doubled up. The doubleQuote( ) method performs this doubling up.

Return Value

The converted string.

Example

```
// Create a DBRowSet from a query in a query file for output.
DBRowSet ds = new DBRowSet(this, "main", "query_filename.gxq",
    "connection_name");

bindLoadValue("NameInput",
    BaseUtils.doubleQuote(request.getParameter("NameInput")));
ds.setQueryName("query_name");
request.setAttribute(ds.getName(),ds);   // Ready for output!
```

Related Topics

DBRowSet class, GX class

includeJSP( )

A convenience method for streaming a JSP file.

Syntax

```
public static void includeJSP(
   HttpServletRequest request,
   HttpServletResponse response,
   String name) throws ServletException, IOException
```

request. The request object.

response. The response object.

name. The name of the JSP or servlet to include.

Usage

The includeJSP( ) method provides a shortcut for streaming a JSP file (or a servlet file). This method also sets the response object's ContentType to "text/html". If an error occurs when including the file, a ServletException is thrown.

initRowSets( )

Initializes RowSet objects specified in a servlet's configuration file.

Syntax

```
public static int initRowSets(
   HttpServletRequest request,
   HttpServletResponse response)
```

request. The request object.

response. The response object.

Usage

Use the initRowSets( ) method to instantiate and initialize all RowSet objects from the metadata in a servlet's configuration file. The initRowSets( ) method also stores the RowSets into the specified request object.

A servlet's configuration file has a "RowSets" entry under the ServletData key. The "RowSets" entry identifies each RowSet that the application designer knew about at design-time.

For each RowSet, the initRowSets( ) method first checks the value of the autoCreate variable. If autoCreate is set to true or is not set, then initRowSets( ) instantiates the RowSet identified by the className variable.

The initRowSets( ) method next initializes each RowSet in either of two ways:

If the RowSet is an instance of IRowSet2, then initRowSets( ) initializes the RowSet by calling the setName( ), setRequest( ), setResponse( ) and initMetaInfo( ) methods of the IRowSet2 interface.

If the RowSet is not an instance of IRowSet2, then the properties in the metadata are assumed to be settable on the RowSet by a call to its setProperty( ) method. The initRowSets( ) method will look up the setProperty( ) method and call it if it's available.

When RowSets are instantiated, the null constructor is used. If a null constructor is not available, the server takes the first constructor defined for that class and calls it with null parameters.

The initRowSets( ) method also creates a new DefaultTemplateMap and adds it into the request object. The DefaultTemplateMap is added as an attribute under the name GX_MAP.

Rule

Do not call initRowSets( ) from an EJB. No provision yet exists for instantiating RowSets from an EJB.

Return Value

Return codes defined in the IServletErrorHandler interface: SUCCESS if the method succeeds; otherwise, FAIL_DONT_STREAM_ERROR or FAIL_STREAM_ERROR.

Related Topics

IRowSet2 interface, IServletErrorHandler interface, DefaultTemplateMap class

log( )

Calls the log( ) method in the AppLogic class and logs an error to the error output.

Syntax 1

Logs a message (type = GXLOG.GXEVENTTYPE_INFORMATION and category = 0).

```
public static int log(
   HttpServletRequest request,
   String message)
```

Syntax 2

Logs an exception instead of a message:

```
public static int log(
   HttpServletRequest request,
   String message,         // message is currently unused
   Throwable t)
```

Syntax 3

Logs an event with a message, specifying the type and category of event:

```
public static int log(
   HttpServletRequest request,
   int type,
   int category,
   String message)
```

message. The message text to log.

request. The request object that contains the pointer to the AppLogic.

t. The Throwable that caused the problem.

type. The type of message to log. Use one of the following variables:

- GXLOG.GXEVENTTYPE_INFORMATION

- GXLOG.GXEVENTTYPE_ERROR
- GXLOG.GXEVENTTYPE_SYSTEM
- GXLOG.GXEVENTTYPE_WARNING

category. User-defined message category. Do not use the range of values reserved for the iPlanet Application Server systems, which is 0 to 65535, inclusive.

Usage

Use log( ) as a convenience method for calling the AppLogic's log( ) method and logging an error to the error output.

In Syntax 2, the request's streamError( ) method is called with a null response object. The IServletErrorHandler object is responsible for converting the exception to a string and then calling:

```
log(HttpServletRequest request, String message)
```

which is the log( ) method defined by Syntax 1.

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

log( ) in the AppLogic class,
IServletErrorHandler interface

DBRowSet class

When you generate JSPs with iPlanet Application Builder (iAB), the resulting JSPs use declarative tags, which support a result-set-oriented view of data rather than a bean-oriented view. To support standard RowSet objects in iAB-generated JSPs, iPlanet Application Server provides the DBRowSet class.

DBRowSet is a declarative-tag-aware extension to the iASRowSet class. Use the methods of DBRowSet in any JSP you have generated with iPlanet Application Builder. iPlanet Application Builder programmers can extend DBRowSet to display formatted, columnar data in a JSP.

DBRowSet can either load queries from a query file or use queries set up by the user. When results (and / or result sets) are requested, the Statement is executed to produce the requested values. After the data is exhausted, the isLast( ) method returns true and the next( ) method returns false.

DBRowSet replaces the DBDataSet and DBStoredProcedure classes from iPlanet Application Builder 6.0.

Package

com.iplanet.server.servlet.extension

Constructors

Syntax 1

Use this syntax for the null constructor:

```
public DBRowSet() throws SQLException
```

Syntax 2

Use the following constructor to allocate a new DBRowSet with a given name and the name of the file from which queries will be loaded.

```
public DBRowSet(
    HttpServletRequest request,
    HttpServletResponse response,
    String name,
    String filename) throws SQLException
```

Syntax 3

Use the following constructor to allocate a new DBRowSet with a given name, the name of the file from which queries will be loaded later, and the name of the connection against which this query will be executed.

```
public DBRowSet(
    HttpServletRequest request,
    HttpServletResponse response,
    String name,
    String filename,
    String connectionName) throws SQLException
```

Syntax 4

Use this constructor like the previous one, but specify a query name as well.

```
public DBRowSet(
    HttpServletRequest request,
    HttpServletResponse response,
    String name,
    String filename,
    String connectionName,
    String queryName) throws SQLException
```

request. The request object.

response. The response object.

name. The name of the RowSet.

filename . The name of the query file from which the queries will be loaded.

connectionName. The default name of the connection to use.

queryName. The name of the query to load from the query file.

SQLException. This exception will be thrown by all constructors whenever the initMetaInfo( ) method throws SQLException.

Methods

This section is divided into four parts:

- Methods That Auto-Execute
- Methods That Auto-Initialize
- Methods That Take a Name Instead of an Index
- Other Methods

Methods That Auto-Execute

The following methods override the corresponding iASRowSet method and execute the RowSet if and only if the RowSet is not already executing. For more information, see either the iPlanet-specific iASRowSet class or the javax.sql.RowSet interface. In particular, see the RowSet execute( ) method or the DBRowSet isExecuted( ) method.

absolute( )	getBytes( )	getShort( )
afterLast( )	getCharacterStream( )	getString( ) — see note below
beforeFirst( )	getClob( )	getTime( )
first( )	getDate( )	getTimeStamp( )
getArray( )	getDouble( )	isLast( )
getAsciiStream( )	getFloat( )	last( )
getBigDecimal( )	getInputFormat( )	moveToCurrentRow( )

getBinaryStream( )	getInt( )	moveToInsertRow( )
getBlob( )	getLong( )	next( )
getBoolean( )	getObject( )	previous( )
getByte( )	getRef( )	relative( )

Note:
The String-parameter version of getString( ) attempts to resolve references to method invocations as well as references to the database columns.

Methods That Auto-Initialize

The following methods override the corresponding iASRowSet method and initialize the RowSet if and only if the RowSet is not already initialized. For more information, see the iPlanet-specific iASRowSet class or the standard RowSet interface. In addition, see the DBRowSet methods init( ) and isInitialized( ).

- execute( )
- getCommand( )
- getStatement( )

Methods That Take a Name Instead of an Index

Queries created with iPlanet Application Builder identify their parameters by name. But JDBC uses query parameters that are identified by an index.

DBRowSet therefore provides a group of “set” methods that have counterparts in the RowSet interface. The only difference is that the DBRowSet method expects a String (for the parameter name), where the corresponding RowSet method expects an integer (for the parameter index). This group of “set” methods is listed below:

setArray( )	setByte( )	setDouble( )	setObject( )
setAsciiStream( )	setBytes( )	setFloat( )	setRef( )
setBigDecimal( )	setCharacterStream( )	setInt( )	setShort( )
setBinaryStream( )	setClob( )	setLong( )	setString( )
setBlob( )	setCommand( )	setName( )	setTime( )
setBoolean( )	setDate( )	setNull( )	setTimestamp( )

In order for your servlets to set parameters on the RowSets, you must translate parameter names to parameter indexes. There are two ways to do this:

- Call any of the set methods listed above, supplying the parameter name as the String value. Each of the listed set methods will in turn call getLoadParameter( ), which returns the index associated with the named query parameter.
- Call the getLoadParameter( ) method directly, supplying the parameter name to convert to a parameter index. Then call the standard RowSet version of the set methods listed above, supplying the parameter index that was previously returned by getLoadParameter( ).

For more information about the previous set methods, see the corresponding methods in the standard javax.sql.RowSet interface.

Other Methods

The remaining DBRowSet methods are described here, with the DBRowSet class. These methods are summarized in the following table:

Method	Description
bindLoadValue( )	Sets a parameter to a particular String value, prior to loading the query.
clearParameters( )	Clears the list of parameters.
getFetchSize( )	Retrieves the value required by the FetchSize hidden field for iAB's "VCR" support.
getLoadParameter( )	Gets the parameter index for a particular parameter in the query.
getName( )	Gets the name for the RowSet.
getNextRowNumber( )	Retrieves one record number beyond the last one that has been printed out.
getQueryFile( )	Gets the name of the file that was used (or will be used) to load queries.
getQueryName( )	Gets the name of the query to be loaded from the query file.
getRowNumber( )	Retrieves the last record number that has been displayed.
getStaticMethodCache( )	Retrieves the method cache for a particular class.
getString( )	Overloads the getString(String) method to use a iASString instead.
init( )	Loads the query.
initMetaInfo( )	Initializes the RowSet to match specific metadata.
isEqualToExpression( )	Evaluates a field specification in an HTML template.

isEqualToValue( )	Evaluates a field specification in an HTML template.
isExecuted( )	Determines whether the RowSet is being executed.
isInitialized( )	Determines whether the RowSet is initialized.
isLastFetchableRecord( )	Returns null if the current record is the last fetchable record; otherwise, it returns the null string.
setExecuted( )	Sets whether the RowSet is being executed.
setInitialized( )	Sets whether the RowSet is initialized.
setName( )	Sets the name for the RowSet.
setQueryFile( )	Sets the name of the file that was used or will be used to load queries.
setQueryName( )	Resets the DBRowSet to access only the given query.
setRequest( )	Sets the request object in which the RowSet will run.
setResponse( )	Sets the response object in which the RowSet will run.

bindLoadValue( )

Sets a parameter to a particular String value, prior to loading the query.

Syntax

```
public int bindLoadValue(
    String destVariable,
    String value)
```

destVariable. The name of the variable to set.

value. The value to set for the variable.

Usage

Use the bindLoadValue( ) method to set parameters when a query is loaded—before the query is executed. In other words, call bindLoadValue( ) before init( ) is called.

Sometimes, it is better to set a parameter at load time rather than at run time. In particular, sometimes you need to replace a parameter to make a SQL statement conform to standard SQL syntax. For example, the following SQL code is nonstandard:

```
select * from emp

where :whereClause
```

If you set the whereClause parameter at load time, this SQL will comply with standard syntax because the parameter is substituted before the command is set on the RowSet.

Return Value

GXE.SUCCESS if the method succeeds.

clearParameters( )

Clears the list of parameters.

Syntax

```
public void clearParameters() throws SQLException
```

Usage

Use clearParameters( ) to clear the list of parameters that were set by the developer.

In general, it is unnecessary to call clearParameters( ) because (typically) you don't re-execute your RowSet.

This method overrides the clearParameters( ) method on the iASRowSet. The iASRowSet, in turn, implements the clearParameters( ) method in javax.sql.RowSet.

Note that clearParameters( ) does not set the RowSet to be uninitialized. As a result, all information that maps parameter names to parameter indexes is lost. To preserve the parameter mapping information, first call the setInitialized( ) method to set the RowSet to be uninitialized. Then call init( ) again to reload the query (and thus the parameter information).

getFetchSize( )

Retrieves the value required by the FetchSize hidden field for iAB's "VCR" support.

Syntax

```
public String getFetchSize(
   iASString max)
```

max. The maximum number to retrieve for the next iteration.

Usage

The FetchSize hidden field supports a “VCR-like” display of records in a window. This type of display is useful when you want to get the next or previous set of records from a query that returns many pages worth of records. For example, the window might contain buttons for displaying the next, previous, first, or last page of data.

Example

The getFetchSize( ) method is invoked from the template engine using a gx-callback such as the following:

```
%gx type=cell id=RowSet.getFetchSize(#max)%%/gx%
```

Return Value

A String representing the value required by FetchSize.

Related Topics

iASString class

getLoadParameter( )

Gets the parameter index for a particular parameter in the query.

Syntax

```
public int getLoadParameter(
   String paramName)
```

paramName. The name of the parameter whose index is to be retrieved.

Usage

Use this method when setting parameters. In iAB, parameters are stored by name, but standard RowSets use parameter indexes. The DBRowSet class provides a group of “set” methods that automatically call getLoadParameter( ) to retrieve the index. For more information, see the section “Methods That Take a Name Instead of an Index.”

After you load the query to define the name-index pairs, you can explicitly call getLoadParameter( ) to retrieve the indices.

If the query is not already loaded (in other words, if the RowSet is not already initialized), then getLoadParameter( ) initializes the RowSet.

Return Value

An integer index associated with a paramater name.

getName( )

Gets the name for the RowSet.

Syntax

```
public String getName()
```

Usage

Use this method to get the name for the RowSet. This is the name that is used to store the RowSet into the attribute list of the Request object.

Related Topics

setName( )

getNextRowNumber( )

Retrieves one record number beyond the last one that has been printed out.

Syntax

```
public String getNextRowNumber(
   iASString unused)
```

unused. Not used.

Usage

This method is useful for populating hidden values for supporting VCR-style display (for example, buttons that display next or previous values).

Example

The getNextRowNumber( ) method is invoked from the template engine, using a gx-callback such as this:

```
%gx type=cell id=RowSet.getNextRowNumber()%%/gx%
```

Return Value

A String.

Related Topics

getRowNumber( )

getQueryFile( )

Gets the name of the file that was used (or will be used) to load queries.

Syntax
```
public String getQueryFile()
```

Usage
Use this method to get the file name that was previously set using setQueryFile( ) or set in the constructor.

Return Value
A String.

Related Topics
setQueryFile( ), getQueryName( )

getQueryName( )
Gets the name of the query to be loaded from the query file.

Syntax
```
public String getQueryName()
```

Usage
Use this method to get the name of the query to be loaded from the query file.

Return Value
A String.

Related Topics
setQueryName( ), getQueryFile( )

getRowNumber( )
Retrieves the last record number that has been displayed.

Syntax
```
public String getRowNumber(
   iASString groupName)
```

groupName. Not used.

Usage
This method is useful for populating table outputs with a row number for each row.

Example
The getRowNumber( ) method is invoked from the template engine, using a gx-callback such as this:

```
%gx type=cell id=RowSet.getRowNumber()%%/gx%
```

Return Value
A String.

Related Topics
getNextRowNumber( )

getStaticMethodCache( )
Retrieves the method cache for a particular class.

Syntax
```
public MethodHash getStaticMethodCache()
```

Usage
In order for RowSet subclasses to be efficiently garbage-collected, each RowSet subclass must override this method to return a different MethodHash. In this way, each class caches only the methods defined in this class or its superclasses without dangling references to the methods of other classes.

Return Value
A MethodHash object.

Related Topics
createMethodCache( ) in the BaseUtils class

getString( )
Overloads the getString(String) method to use a iASString instead.

Syntax
```
public String getString(
   iASString colName) throws SQLException
```

colName. The name of the column, as a iASString.

The javax.sql.RowSet interface has a getString( ) method that takes a String parameter. The getString(iASString) version intercepts the getString(String) version, allowing you to specify a iASString instead.

A SQLException is thrown, as is done in the getString(String) form of the method.

Return Value
A String.

Related Topics
javax.sql.RowSet interface

init( )
Loads the query.

Syntax
```
public void init()
```

Usage
The init( ) method is called implicitly during the execute( ) call.

Related Topics
execute( ), initMetaInfo( )

initMetaInfo( )
Initializes the RowSet to match specific metadata.

Syntax
```
public boolean initMetaInfo(
   OrderedHash metaData) throws SQLException
```

metaData. An OrderedHash of properties. The RowSet will be initialized with these properties.

Usage
The initMetaInfo( ) is used to initialize a RowSet. This method is called by BaseUtils.initRowSets( ) when the RowSet is created.

The metaData parameter specifies the properties stored in a method in the NTV list. The properties specify the query file name, the query name, the connection name, and the input bindings for the query parameters. In this way, the RowSet is initialized so as to match the properties stored in the NTV-list method.

If any subclasses have cached metadata that differs from this RowSet's metadata, then the subclasses should override the method in the NTV list.

Return Value

Returns false if the initialization fails for some reason.

isEqualToExpression( )

Evaluates a field specification in an HTML template.

Syntax

```
public String isEqualToExpression(
   iASString expression)
```

expression. The expression to evaluate.

Usage

This method is similar to isEqualToValue( ). Both methods are intended to be called from the template engine, with the expression containing one equal sign (=) and the lefthand value being a full field specification. In isEqualToValue( ) the righthand value is a constant, whereas in isEqualToExpression( ), the righthand value is another field specification.

The isEqualtoValue( ) method lets you write more standard `if` statements in an HTML template. The isEqualToExpression( ) method is useful for filling a popup list.

Example

The following code fragment creates a data set as a list:

```
MemRowSet ds = new MemRowSet();
ds.setRequest(request);
ds.setResponse(response);
ds.setName("myList");

ds.addListItem("Star Trek", "1");
ds.addListItem("Babylon 5", "2");
ds.addListItem("Red Dwarf", "3");
ds.addListItem("Crusade", "4");
ds.setListSelection("2"); // Babylon 5 is the default selection
request.setAttribute(ds.getName(), ds);
```

To display this list on a web page, you might create an HTML template containing the following code:

```
<SELECT NAME="TVShow">
```

```
%gx type=tile id=myList%
<OPTION VALUE="%gx type=cell id=myList.value% %/gx%"
%gx
type=cell
id=myList.isEqualToExpression(
   myList.value=myList.ListSelection)%SELECTED
%/gx%
>

%gx type=cell id=myList.label%%/gx%
%/gx%
</SELECT>
```

Return Value

Returns null if the expression is true, or "" if the expression is false.

Related Topics

isEqualToValue( ),
MemRowSet class

isEqualToValue( )

Evaluates a field specification in an HTML template.

Syntax

```
public String isEqualToExpression(
   iASString expression)
```

expression. The expression to evaluate.

Usage

This method is similar to isEqualToExpression( ). Both methods are intended to be called from the template engine, with the expression containing one equal sign (=) and the lefthand value being a full field specification. In isEqualToValue( ) the righthand value is a constant, whereas in isEqualToExpression( ), the righthand value is another field specification.

The isEqualtoValue( ) method lets you write more standard `if` statements in an HTML template. The isEqualToExpression( ) method is useful for filling a popup list.

For example, you can use isEqualToValue( ) to determine whether a checkbox should be “CHECKED” or not.

Example

```
<gx type=cell id=isEqualToValue(DataSet.field_name=ALLMINE)>
   HTML specific to ALLMINE
</gx>
```

Return Value

Returns null if the expression is true, or "" if the expression is false.

Related Topics

isEqualToExpression( )

isExecuted( )

Determines whether the RowSet is being executed.

Syntax

```
public boolean isExecuted()
```

Usage

Use isExecuted( ) to determine whether the RowSet is being executed.

Return Value

Returns true if the RowSet is being executed; otherwise, returns false.

Related Topics

setExecuted( )

isInitialized( )

Determines whether the RowSet is initialized.

Syntax

```
public boolean isInitialized()
```

Usage

Use isInitialized( ) to determine whether the RowSet is initialized.

Return Value

Returns true if the RowSet is initialized; otherwise, returns false.

Related Topics

setInitialized( ), init( ), initMetaInfo( )

isLastFetchableRecord( )

Returns null if the current record is the last fetchable record; otherwise, it returns the null string.

Syntax

```
public String isLastFetchableRecord(
   iASString unused)
```

unused. Not used. The default query name is always used.

Usage

This method lets you write JavaScript code that, for example, changes the name of the Next button to be Refresh. In this way, users don't get a blank row when they press the ">" (Next) button at the end of a record set.

Return Value

Returns null if the current record is the last fetchable record; otherwise, it returns the null string.

Example

This method is invoked from the template engine using a gx-callback such as the following:

```
<%gx type=cell id=RowSet.isLastFetchableRecord()%>
<SCRIPT>
datasetFetchCommandNext.name = "datasetFetchCommandRefresh"
</SCRIPT>
<%/gx%>
```

setExecuted( )

Sets whether the RowSet is being executed.

Syntax

```
public void setExecuted(
   boolean executed)
```

executed. The state indicating whether the RowSet is executing or not.

Usage

Use setExecuted( ) to specify the state of RowSet execution. Specify `true` to indicate that the RowSet is being executed, and specify `false` otherwise. You can also use isExecuted( ) to determine whether the RowSet is being executed.

Related Topics

isExecuted( )

setInitialized( )

Sets whether the RowSet is initialized.

Syntax

```
public void setInitialized(
   boolean initialized)
```

initialized. The state indicating whether the RowSet is initialized.

Usage

Use setInitialized( ) to specify the initialization state. Specify `true` to indicate that the RowSet is initialized, which means that the init( ) method has been called. Otherwise, specify `false` . You can also use isInitialized( ) to determine whether the RowSet is initialized.

Related Topics

isInitialized( ), init( ), initMetaInfo( )

setName( )

Sets the name for the RowSet.

Syntax

```
public void setName(
   String name)
```

Usage

Use this method to set the name for the RowSet. You can also set the name using the constructor.

Note that this method also sets the query name—to the same name. If you don't want this to happen, then call setQueryName( ) to reset the query name.

setQueryFile( )

Sets the name of the file that was used or will be used to load queries.

Syntax

```
public int setQueryFile(
   String filename)
```

filename. The name of the file that contains the queries for this RowSet.

Usage

Use setQueryFile( ) to specify the file for loading queries. You can use getQueryFile( ) to retrieve the file name.

Rule

setQueryFile( ) must be called on an uninitialized RowSet.

Return Value

-1 if called on an initialized RowSet.

Related Topics

getQueryFile( ), setQueryName( )

setQueryName( )

Resets the DBRowSet to access only the given query.

Syntax

```
public int setQueryName(
   String query)
```

query. The name of the query that the DBRowSet should load.

Usage

Use setQueryName( ) to reset the DBRowSet to access the specified query. Call setQueryName( ) before executing the DBRowSet.

Return Value

-1 if called on an executed DBRowSet.

Related Topics

getQueryName( ), setQueryFile( )

setRequest( )

Sets the request object in which the RowSet will run.

Syntax

```
public void setRequest(
   HttpServletRequest request)
```

request. The HttpServletRequest object.

Usage

This method is called during construction. The HttpServletRequest is useful when implementing isEqualToExpression( ), where other RowSets may be required.

setResponse( )

Sets the response object in which the RowSet will run.

Syntax

```
public void setResponse(
   HttpServletResponse response)
```

response. The HttpServletResponse object.

Usage

Use setResponse( ) to set the response object in which the RowSet will run.

DefaultHttpSession class

The DefaultHttpSession class is a proxy class for a javax.servlet.http.HttpSession object. When you create subclasses of DefaultHttpSession, these subclasses enable their session objects to conform to the JavaBeans specification (for instance, individual property-getters and property-setters). Both the DefaultHttpSession and user-derived subclasses must have a constructor that takes one argument, the HttpSession.iAB

To retrieve the instance of DefaultHttpSession, you must first ensure that you have correctly configured the SessionInfo section of the application's appInfo.ntv file. In particular, you must set the httpSessionBeanClass attribute to identify the class that acts as proxy for the HttpSession. Assuming that httpSessionBeanClass is set correctly, you can now cast the request to HttpServletRequest2 and call that interface's getHttpSessionBean( ) method.

Note that DefaultHttpSession implements HttpSession. If you want an HttpSession2 object, you must extend the DefaultHttpSession2 class instead.

Although anyone developing a iPlanet Application Server application can use DefaultHttpSession, this class is typically used in components generated by iPlanet Application Builder.

Package

com.iplanet.server.servlet.extension

Constructor

The following constructor is required:

```
public DefaultHttpSession(
    HttpSession baseSession)
```

Use the baseSession parameter to pass in the HttpSession that will serve as the target for all delegated HttpSession methods.

Methods

Most of the methods in the DefaultHttpSession class are delegated from a corresponding method in the HttpSession interface. The following DefaultHttpSession methods are delegated:

- getCreationTime( )
- getId( )
- getLastAccessedTime( )
- getMaxInactiveInterval( )
- getSessionContext( )
- getValue( )
- getValueNames( )
- invalidate( )
- isNew( )
- putValue( )
- removeValue( )
- setMaxInactiveInterval( )

For more information on the previous methods, see the interface javax.servlet.http.HttpSession.

DefaultHttpSession contains the following additional methods:

Method	Description
getHttpSession( )	Returns the delegated implementor of the HttpSession methods.
getString( )	Returns a String value from an HttpSession object.

Method	Description
setHttpSession( )	Sets the delegated implementor of all HttpSession methods.

Related Topics

getHttpSessionBean( ) in the HttpServletRequest2 interface, DefaultHttpSession2 class

getHttpSession( )

Returns the delegated implementor of the HttpSession methods.

Syntax

```
public HttpSession getHttpSession()
```

Return Value

An HttpSession object.

Related Topics

setHttpSession( )

getString( )

Returns a String value from an HttpSession object.

Syntax

```
public String getString(
   String name)
```

name. The name of the string whose value is to be returned.

Usage

The getString( ) method is almost identical to the getValue( ) method. The only difference is that getValue( ) returns an Object value, whereas getString( ) returns a String value.

Return Value

A String.

Related Topics

getValue( ) method in the HttpSession interface, from the Java Servlet API.

setHttpSession( )

Sets the delegated implementor of all HttpSession methods.

Syntax

```
public void setHttpSession(
   HttpSession baseSession)
```

baseSession. The HttpSession object that is delegated as the implementor of all HttpSession methods.

Usage

Use setHttpSession( ) to define the proxy object that implements HttpSession methods. Use getHttpSession( ) to retrieve the object.

Related Topics

getHttpSession( )

DefaultHttpSession2 class

The DefaultHttpSession2 class is a proxy class for an HttpSession2 object. HttpSession2 is an interface of the iPlanet Application Server API, in the package com.iplanet.server.servlet.extension.

When you create subclasses of DefaultHttpSession2, these subclasses enable their session objects to conform to the JavaBeans specification (for instance, individual property-getters and property-setters). Both the DefaultHttpSession2 and user-derived subclasses must have a constructor that takes one argument, the HttpSession. This HttpSession must also be an HttpSession2 object.

In other words, the application must use iPlanet Application Server sessions that implement both HttpSession and HttpSession2. To configure applications to use iPlanet Application Server sessions, set the appropriate session property in the appInfo.ntv file.

To retrieve the instance of DefaultHttpSession2, you must first ensure that you have correctly configured the SessionInfo section of the application's appInfo.ntv file. In particular, you must set the httpSessionBeanClass attribute to identify the class that acts as proxy for the HttpSession2. Assuming that httpSessionBeanClass is set correctly, you can now cast the request to HttpServletRequest2 and call that interface's getHttpSessionBean( ) method.

Note that DefaultHttpSession2 implements HttpSession2. If you want an HttpSession object, you must extend the DefaultHttpSession class instead.

Although anyone developing a iPlanet Application Server application can use DefaultHttpSession2, this class is typically used in components generated by iPlanet Application Builder.

Package

com.iplanet.server.servlet.extension

Constructor

The following constructor is required:

```
public DefaultHttpSession2(
   HttpSession baseSession)
```

Use the baseSession parameter to pass in the Object that will serve as the target for all delegated HttpSession2 and HttpSession methods.

Methods

Most of the methods in the DefaultHttpSession2 class are delegated from a corresponding method in the interface com.iplanet.server.servlet.extension.HttpSession2. The following DefaultHttpSession2 methods are delegated:

- getBytes( )
- getInt( )
- getString( )
- isAuthorized( )
- loginSession( )
- logoutSession( )
- putBytes( )
- putInt( )
- putString( )

For more information on the previous methods, see the HttpSession2 interface in this guide.

DefaultHttpSession2 contains the following additional methods:

Method	Description
getHttpSession( )	Returns the delegated implementor of the HttpSession2 methods.
getString( )	Overrides the invalidate( ) method of DefaultHttpSession to clear out the pointer to the session delegate.

Method	Description
setHttpSession( )	Sets the delegated implementor of all HttpSession2 methods.

Related Topics

getHttpSessionBean( ) in the HttpServletRequest2 interface, DefaultHttpSession class

getHttpSession2( )

Returns the delegated implementor of the HttpSession2 methods.

Syntax

```
public HttpSession2 getHttpSession2()
```

Return Value

An HttpSession2 object.

Related Topics

setHttpSession2( )

invalidate( )

Overrides the invalidate( ) method of DefaultHttpSession to clear out the pointer to the session delegate.

Syntax

```
public void invalidate()
```

Usage

The invalidate( ) method overrides the DefaultHttpSession invalidate( ) method, which in turn is a delegated HttpSession method. Use invalidate( ) to clear out the pointer to the session delegate.

Related Topics

invalidate( ) method in the javax.servlet.http.HttpSession interface

setHttpSession2( )

Sets the delegated implementor of all HttpSession2 methods.

Syntax

```
public void setHttpSession2(
    HttpSession2 baseSession2)
```

baseSession2. The HttpSession2 object that is delegated as the implementor of all HttpSession2 methods.

Usage

Use setHttpSession2( ) to set the HttpSession2 object that acts as the delegated implementor of all HttpSession2 methods. Use getHttpSession2( ) to retrieve the object that was set.

The setHttpSession2( ) method calls setHttpSession( ) if the passed-in baseSession2 parameter is also an HttpSession object.

Related Topics

getHttpSession2( )

DefaultHttpSession2 class

The DefaultHttpSession2 class is a proxy class for an HttpSession2 object. HttpSession2 is an interface of the iPlanet Application Server API, in the package com.iplanet.server.servlet.extension.

When you create subclasses of DefaultHttpSession2, these subclasses enable their session objects to conform to the JavaBeans specification (for instance, individual property-getters and property-setters). Both the DefaultHttpSession2 and user-derived subclasses must have a constructor that takes one argument, the HttpSession. This HttpSession must also be an HttpSession2 object.

In other words, the application must use iPlanet Application Server sessions that implement both HttpSession and HttpSession2. To configure applications to use iPlanet Application Server sessions, set the appropriate session property in the appInfo.ntv file.

To retrieve the instance of DefaultHttpSession2, you must first ensure that you have correctly configured the SessionInfo section of the application's appInfo.ntv file. In particular, you must set the httpSessionBeanClass attribute to identify the class that acts as proxy for the HttpSession2. Assuming that httpSessionBeanClass is set correctly, you can now cast the request to HttpServletRequest2 and call that interface's getHttpSessionBean( ) method.

Note that DefaultHttpSession2 implements HttpSession2. If you want an HttpSession object, you must extend the DefaultHttpSession class instead.

Although anyone developing a iPlanet Application Server application can use DefaultHttpSession2, this class is typically used in components generated by iPlanet Application Builder.

Package

com.iplanet.server.servlet.extension

Constructor

The following constructor is required:

```
public DefaultHttpSession2(
   HttpSession baseSession)
```

Use the baseSession parameter to pass in the Object that will serve as the target for all delegated HttpSession2 and HttpSession methods.

Methods

Most of the methods in the DefaultHttpSession2 class are delegated from a corresponding method in the interface com.iplanet.server.servlet.extension.HttpSession2. The following DefaultHttpSession2 methods are delegated:

- getBytes( )
- getInt( )
- getString( )
- isAuthorized( )
- loginSession( )
- logoutSession( )
- putBytes( )
- putInt( )
- putString( )

For more information on the previous methods, see the HttpSession2 interface in this guide.

DefaultHttpSession2 contains the following additional methods:

Method	Description
getHttpSession( )	Returns the delegated implementor of the HttpSession2 methods.
getString( )	Overrides the invalidate( ) method of DefaultHttpSession to clear out the pointer to the session delegate.

Method	Description
setHttpSession( )	Sets the delegated implementor of all HttpSession2 methods.

Related Topics

getHttpSessionBean( ) in the HttpServletRequest2 interface, DefaultHttpSession class

getHttpSession2( )

Returns the delegated implementor of the HttpSession2 methods.

Syntax

```
public HttpSession2 getHttpSession2()
```

Return Value

An HttpSession2 object.

Related Topics

setHttpSession2( )

invalidate( )

Overrides the invalidate( ) method of DefaultHttpSession to clear out the pointer to the session delegate.

Syntax

```
public void invalidate()
```

Usage

The invalidate( ) method overrides the DefaultHttpSession invalidate( ) method, which in turn is a delegated HttpSession method. Use invalidate( ) to clear out the pointer to the session delegate.

Related Topics

invalidate( ) method in the javax.servlet.http.HttpSession interface

setHttpSession2( )

Sets the delegated implementor of all HttpSession2 methods.

Syntax

```
public void setHttpSession2(
    HttpSession2 baseSession2)
```

baseSession2. The HttpSession2 object that is delegated as the implementor of all HttpSession2 methods.

Usage

Use setHttpSession2( ) to set the HttpSession2 object that acts as the delegated implementor of all HttpSession2 methods. Use getHttpSession2( ) to retrieve the object that was set.

The setHttpSession2( ) method calls setHttpSession( ) if the passed-in baseSession2 parameter is also an HttpSession object.

Related Topics

getHttpSession2( )

GUID class *(deprecated)*

The GUID class is not necessary in the new application model. This class is deprecated and is provided for backward compatiblity only.

The GUID class represents a Globally Unique Identifier, or GUID, which uniquely identifies each application component under iPlanet Application Server. This class provides methods for creating, specifying, and testing GUIDs within application components.

A GUID is a 128-bit hexadecimal number in the following printed format:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Each X is a hexadecimal digit, and the braces and hyphens are required.

GUIDs provide a simple layer of security for your applications because they are not easily human-readable, as in the following example:

```
{E8128C30-6BF8-11cf-96FC-0020AFED9A65}
```

When calling an application component from a web page or from another application component, you must specify a GUID, usually in a string printed format, as shown in the examples in this section. When an application component is called, a request message notifies iPlanet Application Server, which then runs the component.

You must assign and register a unique GUID for each application component you create. For more information about registering GUIDs, see the *Programmer's Guide.*

To instantiate the GUID class, use the Java new keyword, as shown in the following example:

```
GUID myGUID;
myGUID = new GUID();
```

Package

com.kivasoft.types

Methods

Method	Description
GUID( )	Creates a GUID object.
isNull( )	Determines whether the GUID object is null (all zeros).
reset( )	Resets the GUID object to null (all zeros).

Extends

Object.

Example

```
String str;
// Create a unique GUID string
str = runprogram("kguidgen", null, null, null);
if (str != null) {
   //example: str = "{E8128C30-6BF8-11cf-96FC-0020AFED9A65}";
   GUID guid;
   guid = new GUID(str);
   if (!guid.isNull()) {
      return guid;
      }
   }
   return null;
}
```

GUID( )

Creates a GUID object.

Syntax 1

This version, the default constructor method, creates a null GUID containing all zeros.

```
public guid()
```

Syntax 2

This version creates a GUID from a String representation.

```
public GUID(String guidString)
```

guidString. String representing the GUID, using hexadecimal numbers, in the following format:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Return Value

A GUID object containing either all zeros (null) or a 128-bit hexadecimal number, or null for failure (such as the input string or values are not in the correct format).

Example

```
String str;
// Create a unique GUID string
str = runprogram("kguidgen", null, null, null);
if (str != null) {
  // example: str = "{E8128C30-6BF8-11cf-96FC-0020AFED9A65}";
   GUID guid;
   guid = new GUID(str);
   if (!guid.isNull()) {
      return guid;
      }
   }
   return null;
}
```

Related Topics

isNull( ), reset( )

isNull( )

Determines whether the GUID object is null (all zeros).

Syntax

```
public boolean isNull()
```

Usage

Use isNull( ) after calling GUID( ) to determine whether the GUID was created successfully. A null GUID object has the following format:

```
{00000000-0000-0000-0000-000000000000}
```

Return Value

A Boolean true if the specified GUID is null, or false if not.

Example

```
if (!guid.isNull()) {
   return guid;
}
```

Related Topics

GUID( ), reset( )

reset( )

Resets the GUID object to null (all zeros). A null GUID object has the following format:

```
{00000000-0000-0000-0000-000000000000}
```

Syntax

```
public void reset()
```

Example

```
// Reset a GUID to null
guid.reset();
// At this point, guid.isNull() returns true
```

Related Topics

GUID( ), isNull( )

GX class *(deprecated)*

The GX class is deprecated and is provided for backward compatibility only. New applications developed according to the servlet-JSP programming model do not need the functionality provided by the GX class.

The GX class is a utility class for general operations, such as creating a memory buffer or an IValList object. Like the Math and System classes in the Java Class Library, the GX class provides a suite of static methods that can be used anywhere regardless of whether an instance of the class exists or not.

You call the methods in the GX class by using the following convention: GX.*method*. The following example shows how you call CreateBuffer( ), a method in the GX class:

```
IBuffer buff;
buff = GX.CreateBuffer();
```

Package

com.kivasoft.util

Methods

Method	Description
CreateBuffer( )	Creates a new IBuffer object, which represents a block of memory.
CreateBufferFromString( )	Creates a new IBuffer object, which represents a block of memory, and assigns a string value to it.
CreateStreamBuffer( )	Creates a new IStreamBuffer object, which represents a buffer for capturing streamed output during template processing.
CreateTemplateDataBasic( )	Creates a new ITemplateData object, which represents a hierarchical source of data.
CreateTemplateMapBasic( )	Creates a new ITemplateMap object, which represents a mapping between a template field specification and dynamic data used for template processing.
CreateValList( )	Creates a new IValList object, which represents a collection of values.
ProcessOutput( )	Processes the results in an AppLogic's output IValList (vOut) and returns an ITile object from which the caller can extract data.

Method	Description
Release( )	Releases an object from memory.
WaitForOrder( )	Waits for asynchronous operations, such as database queries, to be completed.

CreateBuffer( )

Creates a new IBuffer object, which represents a block of memory.

Syntax

```
public static IBuffer CreateBuffer()
```

Usage

Use CreateBuffer( ) to create IBuffer objects to pass as input arguments to methods, such as put( ) in the TemplateMapBasic class. After you create the IBuffer object, use methods in the IBuffer interface to allocate the memory buffer and to assign a value to it.

Tip

If the buffer is to contain a string, use CreateBufferFromString( ) instead. This method creates the IBuffer object and assigns a string value to it.

Return Value

IBuffer object, or null for failure.

Related Topics

IBuffer Interface (deprecated)

CreateBufferFromString( )

Creates a new IBuffer object, which represents a block of memory, and assigns a string value to it.

Syntax

```
public static IBuffer CreateBufferFromString(
   String str)
```

str. The string to assign to the memory buffer.

Usage

Use CreateBufferFromString( ) to create an IBuffer object that contains a string value. The put( ) method in the TemplateMapBasic class, for example, takes an IBuffer object and a string value as an input argument.

Return Value

IBuffer object, or null for failure.

Example

```
String str = "StateName";
IBuffer buff;
buff = GX.CreateBufferFromString(str);
...
ITemplateMap tmplmap;
tmplmap.put("$STATES", buff);
```

Related Topics

IBuffer Interface (deprecated)

CreateStreamBuffer( )

Deprecated. This method is retained for backward compatibility only.

Creates a new IStream object, which represents a buffer for capturing streamed output during template processing.

Syntax

```
public IStream CreateStreamBuffer(
   IStream stream)
```

stream. Specify null to create a simple stream buffer. Specify another stream buffer to chain two stream buffers.

Usage

Use CreateStreamBuffer( ) to create a stream buffer to pass to evalOutput( ) or evalTemplate( ). The evalOutput( ) and evalTemplate( ) methods merge a template with data from the ITemplateData object and stream the output to the buffer. Use a stream buffer if, for example, your AppLogic needs to manipulate the data before sending it to another AppLogic.

Tip

The IStream object implements the IStreamBuffer interface. To manipulate data in a stream buffer, use the getStreamData( ) method in the IStreamBuffer Interface (deprecated).

Return Value

IStream object, or null for failure.

Related Topics

evalOutput( ) and evalTemplate( ) in the AppLogic class (deprecated)

IStreamBuffer Interface (deprecated)

CreateTemplateDataBasic( )

Deprecated. This method is retained for backward compatibility only.

Creates a new ITemplateData object, which represents a hierarchical source of data.

Syntax

```
public static ITemplateData CreateTemplateDataBasic(
   String str)
```

str. The name of the TemplateDataBasic object.

Usage

Use CreateTemplateDataBasic( ) to create a hierarchical source of data to pass to evalOutput( ) or evalTemplate( ). The evalOutput( ) and evalTemplate( ) methods merge a template with data from the ITemplateData object and stream an output report.

Tip

CreateTemplateDataBasic( ) returns an ITemplateData object. If the AppLogic needs to call a method in the TemplateDataBasic class, cast the ITemplateData object to the TemplateDataBasic class, as shown in the following example:

```
TemplateDataBasic td;

td = (TemplateDataBasic)GX.CreateTemplateDataBasic("td");
```

Return Value

ITemplateData object, or null for failure.

Example

In the following code snippet, two TemplateDataBasic objects are created to store the results from a query to avoid running the same query twice. The two TemplateDataBasic objects are then combined into one and passed to evalTemplate( ) for processing.

```
// Create the TemplateDataBasic objects.
// CreateTemplateDataBasic() returns an ITemplateData
// object, which is cast to TemplateDataBasic
TemplateDataBasic acctsTempDB =
(TemplateDataBasic)GX.CreateTemplateDataBasic("CustAccts");
TemplateDataBasic acctsTempDB2 =
(TemplateDataBasic)GX.CreateTemplateDataBasic("CustAccts2");

String acctDesc;
String acctNum;

// Get the indices of columns in the result set
int acctDescCol = rs.getColumnOrdinal("OBAcctType_acctDesc");
int acctNumCol = rs.getColumnOrdinal("OBAcct_acctNum");

// Loop through the result set and add rows to the
// TemplateDataBasic objects
do
{
   acctDesc = rs.getValueString(acctDescCol);
   acctNum = rs.getValueString(acctNumCol);

   String newRowString = "acctDesc=" + acctDesc + ";" +
                         "acctNum=" + acctNum;

   log(newRowString);
   acctsTempDB.rowAppend(newRowString);
   acctsTempDB2.rowAppend(newRowString);
} while (rs.fetchNext() == 0);

//Create a dummy parent to contain the two template objects
TemplateDataBasic parentTempDB =
(TemplateDataBasic)GX.CreateTemplateDataBasic("Parent");
parentTempDB.rowAppend("Dummy=dummy");// one dummy row
parentTempDB.groupAppend(acctsTempDB);
parentTempDB.groupAppend(acctsTempDB2);

// Merge the template data results with a template
if (evalTemplate("GXApp/OnlineBank/templates/Transfer.html", parentTempDB,
null) != 0)
{
   return handleOBSystemError("Unable to handle template");
}
return (result(null));
```

Related Topics

evalOutput( ) and evalTemplate( ) in the AppLogic class (deprecated)

ITemplateData Interface (deprecated)

TemplateDataBasic class (deprecated)

CreateTemplateMapBasic( )

Deprecated. This method is retained for backward compatibility only.

Creates a new ITemplateMap object, which represents a mapping between a template field specification and dynamic data used for template processing.

Syntax

```
public static ITemplateMap CreateTemplateMapBasic()
```

Usage

Use CreateTemplateMapBasic( ) to create a template map object to pass to evalOutput( ) or evalTemplate( ). A template map object is used to link template fields to calculated values or to source data with a non-matching field name but identically-formatted data.

Tip

CreateTemplateMapBasic( ) returns an ITemplateMap object. If the AppLogic needs to call a method in the TemplateMapBasic class, cast to TemplateMapBasic, as shown in the following example:

```
TemplateMapBasic tmplmap;

tmplmap = (TemplateMapBasic) GX.CreateTemplateMapBasic();
```

Return Value

ITemplateMap object, or null for failure.

Related Topics

evalOutput( ) and evalTemplate( ) in the AppLogic class (deprecated)

ITemplateMap Interface (deprecated)

TemplateMapBasic class (deprecated)

CreateValList( )

Creates a new IValList object, which represents a collection of values.

Syntax

```
public static IValList CreateValList()
```

Usage

Use CreateValList( ) to create IValList objects to pass as input arguments to methods, such as execute( ) in the IPreparedQuery interface. After creating the IValList object, use methods in the IValList interface to set values in the IValList.

Return Value

IValList object, or null for failure.

Example

```
IValList params;
params = GX.CreateValList();
params.setValString(":Make", "Acura")
params.setValString(":Model", "Integra")
...
resultset = pqry.execute(0, params, null, null);
```

Related Topics

IValList interface

ProcessOutput( )

Deprecated. This method is retained for backward compatibility only.

Processes the results in an AppLogic's output IValList (vOut) and returns an ITile object from which the caller can extract data.

Syntax

```
public static ITile ProcessOutput(
   IContext context,
   int flags,
   IValList list)
```

context. The IContext object, which gives the AppLogic access to iPlanet Application Server services. Pass in the AppLogic's context member variable.

flags. Specify 0. Internal use only.

list. The output IValList that contains the results returned by a called AppLogic.

Usage

Use ProcessOutput( ) to process non-HTML results that are returned in the following situation:

1. A client (AppLogic or OCL client application) calls an AppLogic with newRequest( ).
2. Through newRequest( ), the client passes input and output IValLists to the called AppLogic. If the client is an AppLogic, it should specify the value "ocl" for the gx_client_type key in the input IValList. The "ocl" value is assigned automatically for OCL clients.
3. The called AppLogic processes the request and returns results in the output IValList.

ProcessOutput( ) returns the processed data as an ITile object. Data in the ITile object is organized like a hierarchical result set. The client can use methods in the ITile interface to loop through the result set and retrieve values.

Return Value

ITile object, or null for failure.

Example

```
// Call an AppLogic
hr = newRequest(guid, vIn, vOut, 0);
if (hr == GXE.SUCCESS)
{
   // Get the root tile from the output vallist
   ITile roottile = GX.ProcessOutput(null, 0, vOut);
   if (roottile != null)
   {
      String sval;
      ITile ptile;

      //Iterate over all products and print their names
      ptile = roottile.getTileChild("PRODUCTS");
      hr = GXE.SUCCESS;
      System.out.println("Products:");
      while (ptile != null && hr != GXE.FAIL)
      {
         sval = ptile.getTileValue("PRODUCTS.ProdName");
         System.out.println("   " + sval);

         hr = ptile.moveTileNextRecord();
      }
```

Related Topics

newRequest( ) in the AppLogic class (deprecated)

ITile Interface (deprecated)

Release( )

Releases an object from memory.

Syntax

```
public static int Release(
   Object obj)
```

obj. The object to release.

Usage

Use Release( ) to free an object from memory when it is no longer needed. Although Java automatically detects and frees objects that are no longer being used, the Java garbage collector generally runs only at set garbage collection intervals; unused objects might not be freed immediately. Release( ) gives your application more control over object deallocation and system resource usage.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
GX.Release(resultset);
```

WaitForOrder( )

Deprecated. This method is retained for backward compatibility only.

Waits for asynchronous operations, such as database queries, to be completed.

Syntax

```
public static int WaitForOrder(
   IOrder orders[],
   IObject context,
   int timeout);
```

orders. An array of IOrder objects. Each element in the array corresponds to an asynchronous operation.

context. The context object, which is a member variable of the AppLogic class.

timeout. Maximum number of seconds to wait before expiring, if none of the asynchronous queries is finished. Specify 0 to wait forever.

Usage

Use WaitForOrder( ) to wait for one or more asynchronous operations, such as asynchronous database queries, to return the completed results from the database server on which they are submitted. Asynchronous queries that were started with executeQuery( ) in the IDataConn interface might return result sets that are not finished yet. An AppLogic, however, must wait for the result set to be finished before using it.

WaitForOrder( ) waits until one of the following conditions occurs:

- The status of the IOrder object associated with one of the queries changes to GXORDER_STATE.GX_STATE_DONE.
- The specified timeout limit has been exceeded.

Return Value

The index in the orders array of the IOrder that is now completed. WaitForOrder( ) returns a negative number if an error or timeout occurred.

Example

```
class MyAppLogic extends AppLogic
   execute(){
   ...
   IOrder orders[] = new orders[5];
   // fill array with order objects

   int done_index = GX.WaitForOrder(orders, this.context,
                              timeout);
   if (done_index >= 0)
      System.out.println("order finished:" +
                     orders[done_index]);
   else
      System.out.println("error or timeout occurred");
```

Related Topics

IOrder Interface (deprecated)

GXContext class

GXContext is a utility class that enables developers to access iPlanet Application Server services. Most methods of this class are used primarily in the method stubs of Java extensions. For more information about using GXContext while creating extensions, see the *Developer's Guide* that accompanies your version of iPlanet Extension Builder, a separately available product.

In rare situations, EJB and servlet developers may want to call the methods of GXContext, thereby allowing iPlanet Application Server 6.0 applications to include additional iPlanet Application Server features that are not currently supported in the EJB or servlet specifications. For example, the CreateMailbox( ) method supports electronic messaging.

In addition to using GXContext methods, servlet developers can access iPlanet Application Server features in other ways. Servlet developers can use the methods of two iPlanet Application Server-specific interfaces—HttpSession2 or HttpServletRequest2—as well as the methods of AppLogic, a deprecated class.

You call the methods in the GXContext class by using the following convention:

`GXContext.`*`method`*

All of the methods on the GXContext class require the caller to supply an instance of com.kivasoft.IContext. There are many ways in which the context may be obtained, depending on the type of the calling component. Refer to the following examples.

Examples

The following examples describe different ways to obtain a context.

Example 1

From an AppLogic, an instance of this context is available as a member variable `context` within the superclass com.kivasoft.applogic.Applogic. The following code is used from within an AppLogic method:

```
com.kivasoft.IContext kivaContext;

kivaContext = this.context;
```

Example 2

From a servlet, the standard servlet context can be cast to IServerContext, and from there, a com.kivasoft.IContext instance can be obtained. The servlet code would look like this:

```
ServletContext ctx = getServletContext();
```

```
com.iplanet.server.IServerContext sc;


// legal cast within iPlanet Application Server
sc = (com.iplanet.server.IServerContext) ctx;
com.kivasoft.IContext kivaContext = sc.getContext();
```

Example 3

As an alternative to Example 2, a caller can access the underlying AppLogic instance from a servlet and obtain the context there, as described in Example 1 for AppLogics. The servlet code would look like this:

```
HttpServletRequest req;
HttpServletRequest2 req2;
req2 = (HttpServletRequest2) req; // legal cast within iPlanet
Application Server
AppLogic al = req2.getAppLogic();
com.kivasoft.IContext kivaContext;
kivaContext = al.context;
```

Example 4

From a bean, the standard javax.ejb.SessionContext or javax.ejb.EntityContext can be cast to an IServerContext, and from there, a com.kivasoft.IContext instance can be obtained. In a bean, the code would look like this:

```
javax.ejb.SessionContext m_ctx;
.
.
.
com.iplanet.server.IServerContext sc;
// legal cast within iPlanet Application Server
sc = (com.iplanet.server.IServerContext) m_ctx;
com.kivasoft.IContext kivaContext;
kivaContext = sc.getContext();
```

Example 5

From a Java extension, an instance of the context is supplied to the init( ) method of your extension, as shown by the following code:

```
   public int init(IObject obj) {
      com.kivasoft.IContext kivaContext = (com.kivasoft.IContext)
obj;
     .
     .
     .
```

Package

com.kivasoft.dlm

Methods

Method	Description
createMailbox( )	Creates an electronic mailbox object used for communicating with a user's mailbox.
destroySession( )	Deletes a user session.
GetAppEventMgr( )	Retrieves the object for managing application events.
GetObject( )	Retrieves an object from the context.
GetSession( )	Retrieves a session other than the current one.
GetSessionCount( )	Returns the number of sessions in a iPlanet Application Server cluster.
getStateTreeRoot( )	Returns an existing root node of a state tree or creates a new one.
log( )	Writes a message to the server log.
newRequest( )	Calls an AppLogic.
newRequestAsync( )	Calls another AppLogic and runs it asynchronously.

Related Topics

com.kivasoft.applogic.AppLogic class (deprecated),
com.iplanet.server.ICallerContext Interface,
com.kivasoft.IContext Interface,
com.iplanet.server.IServerContext Interface,
com.iplanet.server.servlet.extension.HttpSession2 Interface,
com.iplanet.server.servlet.extension.HttpServletRequest2 Interface

CreateMailbox( )

Creates an electronic mailbox object used for communicating with a user's mailbox.

Syntax

```
public static IMailbox CreateMailbox(
   IContext context,
   String pHost,
   String pUser,
   String pPassword,
   String pUserAddr)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see "Examples."

pHost. Address of POP and SMTP server, such as mail.myOrg.com. If the POP and SMTP servers are running on different hosts, you must use two separate CreateMailbox( ) calls.

pUser. Name of user's POP account, such as jdoe.

pPassword. Password for POP server.

pUserAddr. Return address for outgoing mail, such as john@myOrg.com. Usually the electronic mail address of the user sending the message.

Usage

Use CreateMailbox( ) to set up a mail session for sending and receiving electronic mail messages.

In the Internet electronic mail architecture, different servers are used for incoming and outgoing messages.

- POP (post-office protocol) servers process incoming mail and forward messages to the recipient's mailbox.
- SMTP (simple mail transport protocol) servers forward outgoing mail to the addressee's mail server.

Rules

- The specified user account and password must be valid for the specified POP host name.
- The user address must be valid for the specified SMTP server.

Tip

Once instantiated, use the methods in the IMailbox interface to open and close a mailbox, as well as send and receive mail messages.

Note

In a future release of iPlanet Application Server, this method may be replaced by the JavaMail( ) method.

Return Value

IMailbox object representing a mailbox, or null for failure (such as an invalid parameter).

DestroySession( )

Deletes a user session.

Syntax

```
public static int DestroySession(
   IContext context,
   String pAppName
   String pSessionID
   ISessionIDGen pIDGen)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see “Examples.”

pAppName. Name of the application associated with the session. The application name enables the iPlanet Application Server to determine which calling code has access to the session data.

pSessionID. The session ID to use.

pIDGen. The session ID generation object used to generate session IDs. Specify null.

Usage

To increase security and conserve system resources, use DestroySession( ) to delete a session between a user and the application when the session is no longer required. A caller typically calls DestroySession( ) when the user logs out of an application.

Tip

If the caller set a timeout value for the session when it was created, you need not delete the session explicitly with DestroySession( ). The session is deleted automatically when the timeout expires.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
CreateSession( )

GetAppEventMgr( )
Retrieves the object for managing application events.

Syntax
```
public static IAppEventMgr GetAppEventMgr(
   IContext context)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see “Examples.”

Usage
Use GetAppEventMgr( ) to retrieve an IAppEventMgr object. Through the IAppEventMgr interface, you can create and manage application events. Application event objects define events that are triggered at a specified time or triggered explicitly.

Return Value
IAppEventMgr object, or null for failure.

Related Topics
IAppEventMgr Interface

GetObject( )
Retrieves an object from the context.

Syntax
```
public static Object GetObject(
   IContext context,
   String contextName)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see “Examples.”

contextName. The name of the object in the context. If an object with this name does not exist, null is returned.

Usage

Call GetObject( ) to retrieve a service. For example, call GetObject( ) from an application component in order to access the services of another loaded application component.

Return Value

The returned object, or null for failure.

Example

```
IModuleData modData=GXContext.GetObject(context,
     "com.kivasoft.IModuleData");
```

GetSession( )

Retrieves a session other than the current one.

Syntax

```
public static ISession2 GetSession(
   IContext context,
   int dwFlags,
   String pAppName,
   String pSessionID,
   ISessionIDGen pIDGen)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see “Examples.”

dwFlags. Specify 0 (zero).

pAppName. Name of the application associated with the specified session ID.

pSessionID. The ID of the session you want to retrieve.

pIDGen. The session ID generation object used to generate session IDs. Specify null.

Usage

Call GetSession( ) to retrieve a session other than the current one. For example, you may want to invalidate another session based on some timeout criteria. An application can use GetSession( ) to exert greater control over session management.

Return Value

ISession2 object representing a user session, or null for failure.

GetSessionCount( )

Returns the number of sessions in a iPlanet Application Server cluster.

Syntax

```
public static int GetSessionCount(
   IContext context,
   int dwFlags,
   String appName)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see "Examples."

dwFlags. Unused.

appName. Name of the application in which sessions are being counted.

Usage

Call this method to obtain the number of existing sessions for a given application. If you want your application to limit the number of active sessions, you can call GetSessionCount( ) before creating a new session.

Return Value

The number of sessions.

GetStateTreeRoot( )

Returns an existing root node of a state tree or creates a new one.

Syntax

```
public static IState2 GetStateTreeRoot(
   IContext context,
   int dwFlags,
   String pName)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see "Examples."

dwFlags. Specify one of the following flags or zero to use the default settings.

- GXSTATE.GXSTATE_LOCAL to make the node visible to the local process only.
- GXSTATE.GXSTATE_CLUSTER to make the node visible within the cluster.
- GXSTATE.GXSTATE_DISTRIB, the default, to make the node visible on all servers.

pName. The name of the root node. If a node with this name doesn't already exist, a new node is created.

Usage

Use GetStateTreeRoot( ) to create a state tree. A state tree is a hierarchical data storage mechanism. It is used primarily for storing application data that needs to be distributed across server processes and clusters.

Return Value

IState2 object representing the root node, or null for failure.

Example

The following code shows how to create a state tree and a child node.

```
IState2 tree = GetStateTreeRoot(m_Context,
   GXSTATE.GXSTATE_DISTRIB,
   "Grammy");

if (tree!=null)
{
   IState2 child = tree.getStateChild("Best Female Vocal");
   if (child == null)
   {
      child = tree.createStateChild("Best Female Vocal", 0,
        GXSTATE.GXSTATE_DISTRIB);
```

Log( )

Writes a message to the server log.

Syntax

Logs an event with a message, specifying the type and category of event.

```
public static int Log(
   IContext context,
   int type,
   int category,
   String msg)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see "Examples."

type. Message type. Use one of the following variables:

- GXLOG.GXEVENTTYPE_INFORMATION

- GXLOG.GXEVENTTYPE_ERROR
- GXLOG.GXEVENTTYPE_SYSTEM
- GXLOG.GXEVENTTYPE_WARNING

category. User-defined message category. Do not use the range of values reserved for the iPlanet systems, which is 0 to 65535, inclusive.

msg. Message text to log.

Usage

Use Log( ) for displaying or storing simple messages or for debugging. The output can be directed to the console, to a text file, or to a database table. To direct output, use the iPlanet Application Server Administrator. For more information, see the *Administration and Deployment Guide.*

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
Log(m_Context, GXLOG.GXEVENTTYPE_ERROR", -1,
    "Cannot find table: "+"Shipping");
```

NewRequest( )

Calls an AppLogic.

Syntax 1

Passes in the specified valIn and valOut.

```
public static int NewRequest(
   IContext context,
   String guidSTR,
   IValList vIn,
   IValList vOut,
   int flags)
```

Syntax 2

Use to explicitly specify the location of AppLogic execution.

```
public static int NewRequest(
   IContext context,
   String guidSTR,
   IObject vIn,
   IObject vOut,
   int host,
   int port)
```

Syntax 3

Use to explicitly specify the location of AppLogic execution.

```
public static int NewRequest(
   IContext context,
   String guidSTR,
   IObject vIn,
   IObject vOut,
   int host,
   int port,
   int flags)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see “Examples.”

guidSTR . String GUID or name of the AppLogic to execute.

vIn. IValList object containing input parameters to pass to the called AppLogic.

vOut. IValList object containing result values of the called AppLogic.

host. IP address of the Internet host of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

port. Internet port of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

flags. Specify zero.

Usage

Use NewRequest( ) to call an AppLogic. When you call NewRequest( ), the caller passes to iPlanet Application Server the GUID or name of the AppLogic to execute and, optionally, any input and output parameters.

iPlanet Application Server constructs a request using the parameters specified. iPlanet Application Server then processes the request like any other request, by instantiating the AppLogic and passing in its parameters. The results from the called AppLogic module are returned to the caller.

The AppLogic that NewRequest( ) invokes can do one of the following tasks:

- Process application logic and return result values in the vOut parameter.
- Process application logic and return the resulting data form (such as a report) by streaming the output or by calling result( ).
- Process application logic and return result values in the vOut parameter as well as return the resulting data form (such as a report) by streaming the output or by calling result( ).

If the called AppLogic uses evalOutput( ) to stream results, evalOutput( ) returns HTML results by default. The caller can, however, specify that evalOutput( ) return a non-HTML data stream by setting the gx_client_type key to "ocl" in the input IValList of NewRequest( ). For example:

```
vallist.setValString("gx_client_type", "ocl");
```

Rule

The specified GUID string, input parameters, and output parameters must be valid for the specified AppLogic.

Tips

- The calling code can create new input and output IValLists so as to avoid changing its own input and output IValLists.
- Use NewRequestAsync( ) instead of NewRequest( ) to execute an asynchronous request.
- Called AppLogics might reside on different servers, depending on partitioning and load balancing configurations, might be written in a different language, or might have cached results. The calling code can be unaware or independent of these conditions.
- Using NewRequest( ), you can modularize parts of the application, build dynamic header / footer information and smart reporting templates, and hide complex or confidential business logic in secure submodules or even separate servers.
- Use NewRequest( ) judiciously. Each invoked AppLogic uses a certain amount of communications and server resources.

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Call specified AppLogic and pass parameters
NewRequest(m_Context,
   "{E5CA1000-6EEE-11cf-96FD-0020AFED9A65}",
   paramsToModule, paramsReturned,0);
Example 2
// Use DisplayBasket AppLogic to display the contents
if(NewRequest(m_Context, "DisplayBasket",
        valIn, valOut,0)==0){
   return 0;
   }
else
   return result("Cannot execute DisplayBasket AppLogic");
```

NewRequestAsync( )

Calls another AppLogic and runs it asynchronously.

Syntax 1

Passes in the specified valIn and valOut.

```
public static IOrder NewRequestAsync(
   IContext context,
   String guidSTR,
   IValList vIn,
   IValList vOut,
   int flags)
```

Syntax 2

Use to explicitly specify the location of AppLogic execution.

```
public static IOrder NewRequestAsync(
   IContext context,
   String guidSTR,
   IObject vIn,
   IObject vOut,
   int host,
   int port)
```

Syntax 3

Use to explicitly specify the location of AppLogic execution.

```
public static IOrder NewRequestAsync(
   IContext context,
   String guidSTR,
   IObject vIn,
   IObject vOut,
   int host,
   int port,
   int flags)
```

context. An IContext object, which provides access to iPlanet Application Server services. For information on obtaining a context, see "Examples."

guidSTR . String GUID or name of the AppLogic to execute.

vIn. IValList object containing input parameters to pass to the called AppLogic.

vOut. IValList object containing result values of the called AppLogic.

host. IP address of the Internet host of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

port. Internet port of the iPlanet Application Server where the AppLogic is to be executed. Specify 0 to execute the AppLogic locally.

flags. Specify 0.

Usage

Use NewRequestAsync( ) to call another AppLogic and run it asynchronously. Executing an AppLogic asynchronously is useful if the AppLogic performs a lengthy operation, or if the AppLogic acts as a monitor or remains persistent. For example, an asynchronous AppLogic may perform a lengthy database query to produce a complex result set that it e-mails to a destination address. Another AppLogic module may run continuously and re-index HTML pages every 24 hours.

When NewRequestAsync( ) is called, the caller passes to iPlanet Application Server the GUID of the AppLogic module to execute and, optionally, any input and output parameters.

The iPlanet Application Server constructs a request using the parameters specified and processes it like any other request, by instantiating the AppLogic and passing in its parameters. The results from the called AppLogic module are returned to the calling code.

The AppLogic that NewRequestAsync( ) invokes can do one of the following tasks:

- Process application logic and return result values in the vOut parameter.

- Process application logic and return the resulting data form (such as a report) by streaming the output or by calling result( ).
- Process application logic and return result values in the vOut parameter as well as return the resulting data form (such as a report) by streaming the output or by calling result( ).

Rules

- The specified AppLogic must be accessible to the iPlanet Application Server.
- The specified GUID string, input parameters, and output parameters must be valid for the specified AppLogic module.

Tips

- To get the current status of the request, use the getState( ) method in the returned IOrder object.
- The calling code can use GX.WaitForOrder( ) to wait for multiple asynchronous requests to return.
- Using NewRequestAsync( ), you can modularize parts of the application, build dynamic header/footer information and smart reporting templates, and hide complex or confidential business logic in secure submodules or even separate servers.
- Use NewRequestAsync( ) judiciously. Each invoked AppLogic uses a certain amount of communications and server resources.

Return Value

IOrder object, or null for failure.

Example

```
IOrder  Orders[1];
ULONG     nOrder;
HRESULT hr, ReqResult;

Orders[0] = NewRequestAsync(m_Context, asyncGUIDStr, valIn,
                            valOut, 0);
if (Orders[0] != null)
{
   Log(m_Context, "Successfully invoked async AppLogic\n");

   // wait for async applogic to finish (max 100 seconds)
   hr = waitForOrder(Orders, context, 100);
   if (hr != GXE.SUCCESS)
   {
       return result("Error in executing async request:
```

```
            order wait returned an error");
   }
   else
   {
      getStateIOrder state = Orders[0].getState();
      if (state == null || state.pdwState != GXE.SUCCESS)
         return result("Error in executing async request");
   }
}
else
{
   Log(m_Context,
    GXLOG.GXLOG.GXEVENTTYPE_WARNING,
    -1,
    "Failed to invoke async AppLogic\n");
}
```

GXVAL class *(deprecated)*

GXVAL is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model.

For information about replacing GXVAL functionality in existing applications, see the *Migration Guide*.

The GXVAL class represents a single value of a particular data type. Parameters that are passed to an AppLogic, or results that are retrieved from an AppLogic, are contained in an IValList object that contains one or more GXVAL objects.

Typically, you do not need to use the GXVAL class directly. Generally, you use the getVal***( ) and setVal***( ) suite of methods in the IValList interface to access values in an IValList object. Using the getVal***( ) and setVal***( ) methods in IValList, you can access values of type integer, string, and BLOB in an IValList object. To access more types, such as float or double, use the GXVAL class.

The following example shows how you create a GXVAL object of type double:

```
import com.kivasoft.types.*;
GXVAL v;
v = new GXVAL();
// Set the member variable vt to the double type
v.vt = GXVALTYPE_ENUM.GXVT_R8;
// Call the accessor method for the double type. The member
// variable u holds the value and has accessor methods.
v.u.dblVal(123.45);
```

To create a GXVAL object of a different type, change the GXVALTYPE_ENUM value and the accessor method.

The following table lists the accessor methods and corresponding GXVAL types.

Java type	Accessor method	GXVAL type
char	cVal( )	GXVT_I1
int	iVal( )	GXVT_I2
int	lVal( )	GXVT_I4
float	fltVal( )	GXVT_R4
double	dblVal( )	GXVT_R8
boolean	boolVal( )	GXVT_BOOL
char	bVal( )	GXVT_UI1
int	uiVal( )	GXVT_UI2
int	ulVal( )	GXVT_UI4
java.lang.string	pstrVal( )	GXVT_LPSTR
byte[ ]	pbyteVal( )	GXVT_BLOB
int	pvoidVal( )	GXVT_VOID

Package

com.kivasoft.types

Example

```
import com.kivasoft.types.*;

IValList vallist = GX.CreateValList();
GXVAL x = new GXVAL();
x.vt = GXVALTYPE_ENUM.GXVT_R8;
x.u.dblVal(123.45);
vallist.setVal("test", x);

GXVAL y;
y = vallist.getVal("test");
if (y.vt == GXVALTYPE_ENUM.GXVT_R8 && y.u.dblVal() == 123.45){
   //success
}
```

MemRowSet class

MemRowSet is a subclass of DBRowSet that implements the IListRowSet interface. MemRowSet provides hooks to the in-memory TemplateDataBasic class of iPlanet Application Server. Specifically, MemRowSet wraps the TemplateDataBasic class, adding support for callback methods. For example, a tag such as the following:

```
<gx type=cell id=Jack.callMe(Boisenberry)>
```

would call the method callMe( ) on the rowset named Jack.

Although anyone developing a iPlanet Application Server application can use MemRowSet, this class is typically used in components generated by iPlanet Application Builder.

MemRowSet replaces the DataSet class from iAB 6.0.

Package

com.iplanet.server.servlet.extension

Constructor

The MemRowSet class has two constructors. The first one, the null constructor, is required:

```
public MemRowSet() throws SQLException
```

The second one is a convenience constructor:

```
public MemRowSet(
    HttpServletRequest request,
    HttpServletResponse response,
    String name,
    ITemplateData dataSource) throws SQLException
```

In addition to the request and response objects, the dataSource parameter is the ITemplateData object to use to store data. The name parameter is the MemRowSet name and must correspond with the dataSource's groupName.

Both constructors throw SQLException if it occurs.

Methods

MemRowSet has dozens of methods. Most of them override a corresponding method from one of the following sources:

- iASRowSet class, from package com.iplanet.server.jdbc.
- DBRowSet class, from package com.iplanet.server.servlet.extension.

- RowSet interface, from package javax.sql.
- ResultSet interface, from package java.sql.

The MemRowSet methods are overridden to prevent undesirable behavior. In most cases, a developer should not call the overridden methods, because calling them has no meaning. You might unintentionally call one of these methods on a RowSet object if the RowSet is implemented by MemRowSet, which does not represent a query.

The methods of MemRowSet are listed in the following table. For methods that override a correspondingly named method, only a summary is provided. For new methods of MemRowSet, additional description is also provided after the table. New methods are marked in **boldface** type.

Method	New Behavior of Overridden Method
absolute( )	See separate description below table.
addListItem( )	New method. Adds a new item into a MemRowSet that is being used as an IListRowSet. See separate description below table.
addRow( )	New method. Adds data as a separate row into the data source. See separate description below table.
addValue( )	New method. Queues a value for later addition into the data source. See separate description below table.
afterLast( )	Can never be called. Always throws SQLException.
beforeFirst( )	Can never be called. Always throws SQLException.
cancelRowUpdates( )	Does nothing.
clearWarnings( )	Does nothing.
close( )	Removes the reference to the ITemplateData object.
deleteRow( )	Can never be called. Always throws SQLException.
execute( )	Sets MemRowSet to the “executing” status.
findColumn( )	Returns -1. Columns are not indexed for MemRowSets.
first( )	Can never be called. Always throws SQLException.
getArray( )	Returns null.
getAsciiStream( )	Returns null.
getBigDecimal( )	Returns null.
getBinaryStream( )	Returns null.

Method	New Behavior of Overridden Method
getBlob( )	Returns null.
getBoolean( )	Returns false.
getByte( )	Returns -1.
getBytes( )	Returns null.
getCharacterStream( )	Returns null.
getClob( )	Returns null.
getCommand( )	Returns null.
getConcurrency( )	Returns -1.
getCursorName( )	Returns null.
getDate( )	Returns null.
getDouble( )	Returns -1.
getFetchDirection( )	Returns -1.
getFetchSize( )	Rreturns -1.
getFloat( )	Returns -1.
getInt( )	Returns -1.
getListSelection( )	New method. Gets the list selection of the MemRowSet that is being used as an IListRowSet. See separate description below table.
getLoadParameter( )	Returns 0.
getLong( )	Returns -1.
getMetaData( )	Returns a class that can use ITemplateData objects.
getObject( )	With a String parameter, returns the value returned by getString( ); with an int parameter, returns null.
getQueryName( )	Returns null.
getRef( )	Returns null.
getRow( )	Returns the row number whose contents can currently be examined.
getShort( )	Returns -1.
getStatement( )	Returns null.
getStaticMethodCache( )	Retrieves the method cache for a particular class.

Method	New Behavior of Overridden Method
getString( )	With a String parameter, returns the value returned by its superclass; this form is not overridden. With an int parameter, returns null.
getString( )	With a iASString parameter, either returns the list selection or redirects to the underlying ITemplateData data source.
getTemplateData( )	New method. Gets the data source of the RowSet. See separate description below table.
getTime( )	Returns null.
getTimeStamp( )	Returns null.
getType( )	Returns -1.
getUnicodeStream( )	Returns null.
insertRow( )	Can never be called. Always throws SQLException.
isAfterLast( )	Delegates to the ITemplateData data source. If the RowSet hasn't been accessed, this method returns the value of isEmpty( ) on the ITemplateData. Otherwise, this method returns false.
isBeforeFirst( )	Returns the value of the ITemplateData's isEmpty( ).
isFirst( )	Returns true.
isLast( )	Returns false.
last( )	Must never be called. Always throws SQLException.
moveToCurrentRow( )	Does nothing.
moveToInsertRow( )	Does nothing.
next( )	Called by the template engine to return the next row.
previous( )	Can never be called. Always throws SQLException.
refreshRow( )	Does nothing.
relative( )	Does nothing.
rowDeleted( )	Returns false.
rowInserted( )	Returns false.
rowUpdated( )	Returns false.
setCommand( )	Stubs out uninteresting DBRowSet functionality.
setFetchDirection( )	Does nothing.

Method	New Behavior of Overridden Method
setFetchSize( )	Does nothing.
setListSelection( )	New method. Sets the list selection on the MemRowSet that is being used as an IListRowSet. See separate description below table.
setName( )	Does nothing.
setQueryName( )	Stubs out uninteresting DBRowSet functionality.
setTemplateData( )	New method. Sets the data source to the provided TemplateDataBasic. See separate description below table.
setupQuery( )	Stubs out uninteresting DBRowSet functionality.
updateAsciiStream( )	Does nothing.
updateBigDecimal( )	Does nothing.
updateBinaryStream( )	Does nothing.
updateBoolean( )	Does nothing.
updateByte( )	Does nothing.
updateBytes( )	Does nothing.
updateCharacterStream( )	Does nothing.
updateDate( )	Does nothing.
updateDouble( )	Does nothing.
updateFloat( )	Does nothing.
updateInt( )	Does nothing.
updateLong( )	Does nothing.
updateNull( )	Does nothing.
updateObject( )	Does nothing.
updateShort( )	Does nothing.
updateString( )	Does nothing.
updateTime( )	Does nothing.
updateTimestamp( )	Does nothing.
wasNull( )	Returns false.

addListItem( )

Adds a new item into a MemRowSet that is being used as an IListRowSet.

Syntax

```
public void addListItem(
   String label,
   String value)
```

label. The name of the item as it appears in the list.

value. The value assigned to the label.

Usage

When working with an IListRowSet, use addListItem( ) with setListSelection( ) to create a selection list or a group of radio buttons.

Example

The following code fragment creates a data set as a list:

```
MemRowSet ds = new MemRowSet();
ds.setRequest(req);
ds.setResponse(res);
ds.setName("myList");
ds.addListItem("Star Trek", "1");
ds.addListItem("Babylon 5", "2");
ds.addListItem("Red Dwarf", "3");
ds.addListItem("Crusade", "4");
ds.setListSelection("2"); // Babylon 5 is the default selection
req.setAttribute(ds.getName(),ds);
```

To display this list on a web page, you might create an HTML template containing the following code:

```
<SELECT NAME="TVShow">
%gx type=tile id=myList%
<OPTION VALUE="%gx type=cell id=myList.value% %/gx%"
%gx
type=cell
id=myList.isEqualToExpression(
    myList.value=myList.ListSelection)%SELECTED
%/gx%
>
```

```
%gx type=cell id=myList.label%%/gx%
%/gx%
</SELECT>
```

Related Topics
setListSelection( )

addRow( )

Adds data as a separate row into the data source.

Syntax
```
public int addRow()
```

Usage
Use addRow( ) to add a new row to a data set. The values of the row must have been previously added using addValue( ); otherwise, no row is added.

Example

```
MemRowSet ds = new MemRowSet();
ds.setRequest(req);
ds.setResponse(res);
ds.setName("tryMe");
ds.addValue("col1", "Hello");
ds.addValue("col2", "World");
ds.addRow();
ds.addValue("col1", "GoodBye");
ds.addValue("col2", "World");
ds.addRow();
req.setAttribute(ds.getName(),ds);
```

The previous code, along with an HTML template containing the following:

```
<gx type=tile id=tryMe>
<gx type=cell id=tryMe.col1></gx> <gx type=cell id=tryMe.col2><%gx>
<BR>
</gx>
```

would produce:

```
Hello World
Goodbye World
```

Return Value
Unused. Always 0.

Related Topics
addValue( ), addListItem( )

addValue( )
Queues a value for later addition into the data source.

Syntax
```
public int addValue(
   String name,
   String value)
```

name. The column name of the value.

value. The value to add to the data source.

Usage
In-memory data sources start out without any data. Use a series of addValue( ) and addRow( ) calls to populate a data source with values.

Note that data is not actually added to the data source until addRow() is called.

Example
See addRow( ) for an example.

Return Value
Unused. Always 0.

Related Topics
addRow( )

getListSelection( )
Gets the list selection of the MemRowSet that is being used as an IListRowSet.

Syntax
```
public String getListSelection(
   String value)
```

value. The value to add to the data source.

Usage
Use getListSelection( ) to obtain the list selection previously set by a call to setListSelection( ).

Return Value
A String representing the list selection.

Related Topics
setListSelection( )

getTemplateData( )
Gets the data source of the RowSet.

Syntax
```
public ITemplateData getTemplateData()
```

Usage
Use getTemplateData( ) to obtain the ITemplateData that serves as the data source of the RowSet. This ITemplateData object was created in either of two ways:

- implicitly, as a result of calling addRow( ).
- by calling setTemplateData( ).

Return Value
An ITemplateData representing the data source.

Related Topics
setTemplateData( )

setListSelection( )
Sets the list selection on the MemRowSet that is being used as an IListRowSet.

Syntax
```
public void setListSelection(
   String value)
```

value. The value to be set for the list selection.

Usage

When working with an IListRowSet, use addListItem( ) with setListSelection( ) to create a selection list or a group of radio buttons. Use getListSelection( ) to obtain the list selection previously set by a call to setListSelection( ).

Example

See addListItem( ) for an example.

Related Topics

addListItem( ), getListSelection( )

setTemplateData( )

Sets the data source to the provided TemplateDataBasic.

Creates an ITemplateData object on a data set.

Syntax

```
public void setTemplateData(
   ITemplateData td)
```

td. The name of the ITemplateData object to create.

Usage

After you call setTemplateData( ) with a specified ITemplateData object, data requests such as calls to getString( ) and next( ) will be routed through the passed-in ITemplateData. Therefore, if you already have an ITemplateData (for example, from an extension), you can wrap a data set around it by creating a MemRowSet and calling setTemplateData( ) on the MemRowSet to pass in the ITemplateData.

Furthermore, if the ITemplateData is also a TemplateDataBasic, then calls to addRow( ) and setHint( ) will succeed. However, if the ITemplateData is not a TemplateDataBasic, calls to add a row or set the hint will fail, returning a value of -1.

Related Topics

getTemplateData( )

iAS6.0RowSet class

A RowSet is an object that encapsulates a set of rows retrieved from a database or other tabular data store, such as a spreadsheet. To implement a RowSet, your code must import javax.sql, and implement the RowSet interface. RowSet extends the java.sql.ResultSet interface, permitting it to act as a JavaBeans component.

Because a RowSet is a JavaBean, you can implement events and event listeners for the RowSet, and you can set properties on the RowSet. Furthermore, because RowSet is an extension of ResultSet, you can iterate through a RowSet just as you would iterate through a result set.

You fill a RowSet by calling the RowSet.execute( ) method. The execute( ) method uses property values to determine the data source and retrieve data. The properties you decide to set and examine depend on the implementation of RowSet you invoke.

Package

com.iplanetiplanet.server.jdbc

Constructors

In a iPlanet Application Server application, you can create a RowSet in either of two ways:

- You can create iPlanet Application Server-dependent code by calling new on the iASRowSet class:

```
iASRowSet rs = new iASRowSet();
```

- You can create iPlanet Application Server-independent code. To do so, call new on the InitialContext class. Then, using this InitialContext, look up the class that implements the RowSet interface:

```
InitialContext ctx = new InitialContext();
RowSet rs = (javax.sql.RowSet) ctx.lookup("javax.sql.RowSet");
```

  The ctx.lookup( ) call returns an instance of the class that is bound to the implementation of javax.sql.RowSet. In this case, the class that is bound is iASRowSet.

Methods

This section provides details about what iASRowSet implements.

iASRowSet is a class that extends ResultSet, so the iASRowSet methods are inherited from the ResultSet object.

In addition, iASRowSet does the following:

- implements, in whole or in part, these four standard interfaces:
  - javax.sql.RowSet
  - javax.sql.RowSetReader
  - javax.sql.RowSetWriter
  - javax.sql.RowSetInternal
- implements many methods available in CachedRowSet, a class provided by Sun.
- implements loadQuery( ), a method of the SqlUtil class in the com.kivasoft.util package.

RowSet interface

The RowSet interface is fully supported except as noted in the following table:

Method	Argument	Action	Reason
setReadOnly( )	false	throws SQLException	iASRowSet is already read-only.
setType()	TYPE_SCROLL_ SENSITIVE	throws SQLException	Argument is not supported.
setConcurrency( )	CONCUR_ UPDATABLE	throws SQLException	iASRowSet is read-only.
addRowSetListener( )	any	None	Not supported.
removeRowSetListener( )	any	None	Not supported.
setNull( )	any type name	ignores argument	Not supported.
setTypeMap( )	java.util.Map	None	Map is a JDK 1.2 feature, so is not supported.

RowSetReader interface

iASRowSet provides a full implementation of the RowSetReader interface. As an alternative to one of the RowSetReader methods, you can use the setReaderMethod( ) from the CachedRowSet class. The argument of setReaderMethod( ) must nonetheless implement the RowSetReader interface.

RowSetWriter interface

Although iASRowSet is read-only, it implements the RowSetWriter interface for future use. The only method of RowSetWriter, writeData( ), throws SQLException when called, indicating that this method is not supported.

RowSetInternal interface

This internal interface is called by a method of the RowSetReader interface to retrieve information about the RowSet. RowSetInternal has a single method, getOriginalRow( ), which returns the entire result set instead of a single row.

CachedRowSet class

Sun provides a rowset class named CachedRowSet, which is an implementation of the RowSet interface. CachedRowSet lets you retrieve data from a data source, then detach from the data source while you examine (and modify) the data. A cached rowset keeps track of not only the original data retrieved, but also any data changes your application made. If the application trys to update the original data source, the rowset is reconnected to the data source, and only those rows that have changed are merged back into the database.

iASRowSet implements many of the methods in the CachedRowSet class because they're useful. However, iASRowSet does not implement all the methods. Unimplemented methods throw SQLException if called. The following CachedRowSet methods throw SQLException because iASRowSet is read-only:

- acceptChanges( )
- cancelRowDelete( )
- cancelRowInsert( )
- getShowDeleted( )
- restoreOriginal( )
- setShowDeleted( )
- setWriter( )

The toCollection( ) method throws SQLException because collections are a JDK 1.2 feature.

SqlUtil class

The SqlUtil class is part of the iPlanet Application Server Foundation Class Library. This class contains loadQuery( ), a method that supports rowsets. The loadQuery( ) method can be used on the RowSet interface in place of the setCommand( ) method.

For example:

```
import com.kivasoft.util.SqlUtil;

...

SqlUtil.loadQuery(rowset, pathname, queryname, params);
```

This version of loadQuery( ) reads the queryname from the file specified by pathname. The method then replaces parameters according to params (an IValList object) and sets the query command as the RowSet query command.

Related Topics

RowSet, RowSetInternal, RowSetReader, or RowSetWriter interfaces in the javax.sql package,
java.sql.ResultSet interface,
com.kivasoft.util.SqlUtil class

iASString class

The iASString class shadows the java.lang.String class.

Strings are constant. However, iASStrings are not. Their value can be changed, and they provide a simple method to change which part of a String to operate on. In this way, you can section off parts of a String without the overhead of creating a new String. In addition, iASString conforms to the method signatures provided for the String class.

Although anyone developing a iPlanet Application Server application can use iASString, this class is typically used in components generated by iPlanet Application Builder.

Package

com.iplanet.server.servlet.extension

Constructors

The following constructors are available:

Constructor	Type of iASString Created
`public iASString()`	Allocates a new iASString containing no characters. Using this constructor is not recommended.
`public iASString( String value)`	Allocates a new iASString that shadows the passed-in String. The iASString will be identical to the shadow string, except that it is changeable.
`public iASString( iASString value)`	Allocates a new iASString that is identical to the current substring of the passed-in iASString.

Methods

iASString and String are almost identical. Every instance method on java.lang.String has a corresponding, functionally identical method on iASString. Every instance method on String that takes a String as a parameter is overloaded in iASString to take a String or a iASString.

The following additional methods are in iASString but not in String.

Method	Description
equals( )	Compares this iASString to the specified parameter.
makeSubstring( )	Modifies this iASString to point to a substring of shadow.
setValue( )	Resets the value of this iASString to shadow the specified String.

equals( )

Compares this iASString to the specified parameter.

Syntax 1

```
public boolean equals(
   Object anObject)
```

Syntax 2

```
public boolean equals(
   String val)
```

Syntax 3

```
public boolean equals(
   iASString string)
```

Usage

The equals( ) method compares this iASString to the specified parameter, either an Object, a String, or an iASString.

This method returns true if the parameter is not null and if it meets the following conditions:

- When specified as an Object (Syntax 1), the parameter must be either:
  - a String object that represents the same sequence of characters as this object
  - an iASString that shadows a substring of a String representing the same sequence of characters as this object
- When specified as String (Syntax 2), the parameter must be a String object that represents the same sequence of characters as this object
- When specified as an iASString (Syntax 3), the parameter must be an iASString that shadows a substring of a String representing the same sequence of characters as this object

The equals( ) method allows the compiler to bind to this method when the class of the object is known. This binding allows for removal of the code to check the class of the incoming object.

Return Value

Returns true if the previous conditions are met; otherwise, returns false.

makeSubstring( )

Modifies this iASString to point to a substring of shadow.

Syntax

```
public void makeSubstring(
   int beginIndex)
```

Syntax

```
public void makeSubstring(
   int beginIndex,
   int endIndex)
```

beginIndex. The beginning index. If the iASString is already pointing at a substring, then this index can be negative, indicating that the beginning of the substring will occur earlier. If beginIndex is out of range, this method throws StringIndexOutOfBoundsException.

endIndex. The ending index, excluding itself.

Usage

Use makeSubstring( ) to modify this iASString to point at a substring of shadow. The substring begins at the specified index, which is the offset from the current substring being shadowed by the iASString.

If no endIndex is specified, the substring extends to the end of the currently shadowed substring. If endIndex is specified, the substring extends to the character located at endIndex - 1 of the newly shadowed substring.

setValue( )

Resets the value of this iASString to shadow the specified String.

Syntax

```
public void setValue(
    String val)
```

val. The String to shadow.

Usage

Use setValue( ) to reset the value of this iASString to shadow the specified String.

iASStringBuffer class

iASStringBuffer is a subclass of Object that resembles the java.lang.StringBuffer class (an object that manages a mutable string). Unlike java.lang.StringBuffer, none of iASStringBuffer's methods are synchronized, so performance is significantly increased.

iASStringBuffer also has additional methods that allow it to be more easily modified than the standard StringBuffer. Unfortunately, it is slower than StringBuffer when converting to a string.

Although anyone developing a iPlanet Application Server application can use iASStringBuffer, this class is typically used in components generated by iPlanet Application Builder.

iASStringBuffer replaces the FastStringBuffer class from iAB 6.0.

Package

com.iplanet.server.servlet.extension

Constructors

The following constructors are available:

Constructor	Type of iASStringBuffer Created
`public iASStringBuffer()`	Contains the empty string.
`public iASStringBuffer (String aString)`	Contains the characters in aString. The buffer is copied, not referenced.
`public iASStringBuffer( String aString, int start, int end)`	Contains the range of characters in aString from the index start to end, with end excluded.
`public iASStringBuffer( char aChar)`	Contains the character aChar.
`public iASStringBuffer( int size)`	Has the given length.
`public iASStringBuffer (iASStringBuffer nsb)`	Contains another iASStringBuffer's contents. The buffer is copied, not referenced.
`public iASStringBuffer (char buff[])`	Contains the specified character array. The buffer is copied, not referenced.

Methods

Method	Description
append( )	Appends a character or string to the iASStringBuffer.
charArray( )	Returns the iASStringBuffer's character array.
charAt( )	Returns the character at the specified index.
ensureCapacity( )	Makes sure that the StringBuffer has enough space to contain "newCapacity" characters.
equals( )	Compares the contents of this iASStringBuffer with the contents of another iASStringBuffer.

Method	Description
getChars( )	Copies characters from the iASStringBuffer to the specified destination array.
indexOf( )	Returns the index of the first occurrence of the specified character in the iASStringBuffer.
insert( )	Inserts a character or string into the iASStringBuffer.
length( )	Returns the number of characters in the iASStringBuffer.
moveChars( )	Moves the tail of the string, by removing characters.
removeCharAt( )	Removes the character at the specified index.
replace( )	Replaces the given range with a string, or removes the given range.
setCharAt( )	Sets the value of a character at the specified index.
setLength( )	Truncates the iASStringBuffer to the specified number of characters.
setStringValue( )	Sets the contents of the iASStringBuffer to the characters in the specified string.
tabOrSpaceAt( )	Determines whether a tab or space exists at the specified index.
toString( )	Returns the String for the iASStringBuffer's contents.
trimToSize( )	Truncates the internal character array to match the size of the contained string.
truncateToLength( )	Same as the trimToSize( ) method.

append( )

Appends a character or string to the iASStringBuffer.

Syntax 1

```
public void append(
   char aChar)
```

Syntax 2

```
public void append(
   char str[])
```

Syntax 3

```
public void append(
   String aString)
```

Syntax 4

```
public void append(
   char str[],
   int offset,
   int length)
```

aChar. The character to append.

aString. The string to append.

str. The characters to append.

offset. The index of the first character to append. If the offset is invalid, this method throws StringIndexOutOfBoundsException

length. The number of characters to append.

Usage

The first three forms of append( ) let you append a character or a string to the iASStringBuffer. In the last form of this method, characters in the specified character array, str, are appended in order, starting at the specified offset. The length of the iASStringBuffer increases by the specified length.

This method returns the modified iASStringBuffer.

charArray( )

Returns the iASStringBuffer's character array.

Syntax

```
public char[] charArray()
```

Usage

Use this method whenever you need to access the character array. However, do not modify this array yourself.

For example, suppose you want to draw the contents of the iASStringBuffer. You can do this by passing the array to the graphic's drawString( ) method that takes a character array, rather than converting the StringBuffer to a String.

Tip

You may want to call trimToSize() first; otherwise, some characters in the array will be out of the “valid” range for the StringBuffer.

charAt( )

Returns the character at the specified index.

Syntax

```
public char charAt(
   int index)
```

index. The position of the character.

Usage

If the index is invalid, this method throws StringIndexOutOfBoundsException.

ensureCapacity( )

Makes sure that the StringBuffer has enough space to contain “newCapacity” characters.

Syntax

```
public void ensureCapacity(
   int newCapacity)
```

Usage

Called internally.

equals( )

Compares the contents of this iASStringBuffer with the contents of another iASStringBuffer.

Syntax

```
public boolean equals(
   iASStringBuffer value)
```

Usage

Specify a value to compare.

getChars( )

Copies characters from the iASStringBuffer to the specified destination array.

Syntax

```
public void getChars(
   int srcBegin,
   int srcEnd,
   char dst[],
   int dstBegin)
```

srcBegin. The offset in the source iASStringBuffer at which copying begins.

srcEnd. The offset in the source iASStringBuffer at which copying ends.

dst. The destination array, to which data is copied.

dstBegin. The offset at which copying into the destination array begins.

Usage

Use getChars( ) to copy characters to an array. The first character to be copied is at the index srcBegin, and the last character to be copied is at srcEnd - 1. The total number of characters to be copied is srcEnd - srcBegin.

These characters are copied into a subarray of the specified destination array, dst. This subarray begins at the index dstBegin and ends at the following index:

```
dstBegin + (srcEnd - srcBegin) - 1
```

If any index is invalid, this method throws StringIndexOutOfBoundsException.

indexOf( )

Returns the index of the first occurrence of the specified character in the iASStringBuffer.

Syntax 1

```
public int indexOf(
   char aChar,
   int offset)
```

Syntax 2

```
public int indexOf(
   char aChar
```

aChar. The character whose position you are looking for.

offset. The starting position for searching in the iASStringBuffer.

Usage

You can use indexOf( ) in two forms. The first form starts at the specified offset in iASStringBuffer and returns the index of the specified character's first occurrence. If the offset is invalid, this method throws StringIndexOutOfBoundsException.

The second form is similar, except that is starts at the beginning of the iASStringBuffer and is equivalent to the following code:

```
indexOf(aChar, 0);
```

insert( )

Inserts a character or string into the iASStringBuffer.

Syntax 1

```
public void insert(
   int index,
   char aChar)
```

Syntax 2

```
public void insert(
   int index,
   String aString)
```

Syntax 3

```
public void insert(
   int offset,
   char str[])
```

index. The position in the iASStringBuffer.

aChar. The character to insert.

aString. The string to insert.

offset. The starting position for searching in the iASStringBuffer.

str. The character array to insert.

Usage

The first two forms of the insert( ) method insert either a character or a string at the specified index.

If the index equals or exceeds the number of characters in the buffer, then this method appends the item. If the index is invalid, this method throws StringIndexOutOfBoundsException.

In the last form of this method, characters in the specified character array, str, are appended in order, starting at the specified offset. The length of the iASStringBuffer increases by the length of the argument.

This method returns the modified iASStringBuffer.

length( )

Returns the number of characters in the iASStringBuffer.

Syntax

```
public int length()
```

moveChars( )

Moves the tail of the string, by removing characters.

Syntax

```
public void moveChars(
   int fromIndex,
   int toIndex)
```

fromIndex. Position of the first character to move. The affected characters range from this position to the end of the iASStringBuffer.

toIndex. Position to move the affected characters.

Usage

If characters are moved to an index within the existing string, the previous characters are overwritten by the move.

removeCharAt( )

Removes the character at the specified index.

Syntax

```
public void removeCharAt(
   int index)
```

index. The index of the character to remove.

Usage

If the index is invalid, this method throws StringIndexOutOfBoundsException.

replace( )

Replaces the given range with a string, or removes the given range.

Syntax

```
public void replace(
   int begin,
   int end,
   String value)
```

begin. The index of the beginning of the range.

end. The index of the end of the range.

value. The string that replaces the range. If null, the range is removed.

Usage

Use replace( ) to remove or replace characters in a iASStringBuffer. If either of the range indexes is invalid, this method throws StringIndexOutOfBoundsException.

setCharAt( )

Sets the value of a character at the specified index.

Syntax

```
public void setCharAt(
   int index,
   char ch)
```

index. The position of the character to set.

ch. The value to set the character to.

Usage

If the index is invalid, this method throws StringIndexOutOfBoundsException.

setLength( )

Truncates the iASStringBuffer to the specified number of characters.

Syntax

```
public void setLength(
   int length)
```

length. Number of characters to remain in the truncated iASStringBuffer. If the specified length is invalid, this method does nothing.

setStringValue( )

Sets the contents of the iASStringBuffer to the characters in the specified string.

Syntax

```
public void setStringValue(
   String aString)
```

aString. The string to use as the new contents of iASStringBuffer.

tabOrSpaceAt( )

Determines whether a tab or space exists at the specified index.

Syntax

```
public boolean tabOrSpaceAt(
   int index)
```

index. The position in the iASStringBuffer.

Usage

The tabOrSpaceAt( ) method returns true if the specified index contains a tab or a space; otherwise, this method returns false. If the index is invalid, this method throws StringIndexOutOfBoundsException.

toString( )

Returns the String for the iASStringBuffer's contents.

Syntax

```
public String toString()
```

Usage

This method overrides toString( ) in the Object class.

trimToSize( )

Truncates the internal character array to match the size of the contained string.

Syntax

```
public void trimToSize()
```

Usage

Use trimToSize( ) when more space than necessary is allocated to the iASStringBuffer, and you want to free this space for other uses.

truncateToLength( )

Same as the trimToSize( ) method.

Syntax

```
public void truncateToLength(
    int aLength)
```

Session2 class *(deprecated)*

The Session2 class is deprecated and is provided for backward compatibility only. New applications should use the methods provided by the standard javax.servlet.http.HttpSession interface. In addition, iPlanet Application Server provides the HttpSession2 interface, an extension to HttpSession that supports applications that must call AppLogics.

The Session2 class is designed to help you implement a custom session class if your application requires additional session functionality. To create a custom session class, subclass the Session2 class, then define new methods. Your subclass can, for example, define accessor methods to read and write information specific to your session. An online shopping application, for example, might require specialized methods, such as AddItemToCart( ), to track shopping items per user session.

When you subclass the Session2 class, you must do the following:

- Override the createSession( ) and getSession( ) methods in the AppLogic class. In these methods, you can invoke the base AppLogic methods to obtain an ISession2 object, and construct your own session class by passing in this object, as shown in the following example:

```
ISession2 s = super.createSession(flags, timeout, appname, sessionid, idgen);

if (s!= null)
    sess = new MySession(s);

return s;
```

- Pass in the ISession2 interface in the subclass constructor, as shown in the following example:

```
public class MySession extends Session2
```

```
{
    public MySession(ISession2 sess)
```

Because the Session2 class delegates the implementation of methods in the ISession2 interface to the object passed to its constructor, you don't have to implement every method of that interface in your subclass. You need only define the methods you want to add.

Package

com.kivasoft.session

Related Topics

ISession2 Interface (deprecated)

SqlUtil class

The SqlUtil class contains helper methods for two main purposes:

- to load queries from query files (.gxq files)
- to create or initialize JDBC classes for database access.

At present, this class contains only one helper method, loadQuery( ). This method has six syntax variants.

Although anyone developing a iPlanet Application Server application can use SqlUtil, this class is typically used in components generated by iPlanet Application Builder.

Package

com.kivasoft.util

Variables

Variable	Description
minLocChar	Lower end set of characters that are considered part of a variable name, in addition to a-zA-Z0-9.
maxLocChar	Upper end set of characters that are considered part of a variable name, in addition to a-zA-Z0-9.

Constructor

SqlUtil extends the Object class and should never be instantiated. The SqlUtil constructor is as follows:

```
public SqlUtil()
```

Methods

Method	Description
loadQuery( )	Loads a query.

Examples

Example 1

The following code shows how you might use a SELECT query:

```
try
{
    int colIndex;
    int x;
    long y;
    iASRowSet rowSet = new com.iplanet.server.jdbc.iASRowSet();

    // Initialize connection info
    rowSet.setDataSourceName("jdbc/nsample");

    // permanent parameter substitutions in the SQL statement.
    IValList param = GX.CreateValList();
    param.setValString("REGION", "WEST");
    param.setValInt("POPULATION", 100000);

    SqlUtil.loadQuery(rowSet, "state.gxq", "STATES_select1",
param);

    // temporary parameter substitutions.
    rowSet.setString(param.getValInt("STATE"), "CA");
    rowSet.execute();

    // get the return values.
    colIndex = 1;
    x = rowSet.getInt(colIndex++);
    y = rowSet.getLong(colIndex++);

    rowSet.clearParameters();
    // temporary parameter substitutions.
```

```
try
    rowSet.setString(param.getValInt("STATE"), "TX");
    rowSet.execute();

    // get the return values.
    colIndex = 1;
    x = rowSet.getInt(colIndex++);
    y = rowSet.getLong(colIndex++);

    rowSet.close();
}
catch (SQLException err)
{
}
```

Example 2

The following code shows how you might use an UPDATE query:

```
try
{
    String url = "jdbc:iplanet:ODBC:nsample"
    java.util.Properties props = new java.util.Properties();
    props.put("DRIVER", "ODBC");
    props.put("DSN", "nsample");
    props.put("DB", "nsample");
    props.put("user", "iplanet");
    props.put("password", "iplanet");
    Connection conn = DriverManager.getDriver(url).connect(url,
props);

    // permanent parameter substitutions in the SQL statement.
    IValList param = GX.CreateValList();
    param.setValString("REGION", "WEST");
    param.setValInt("POPULATION", 100000);

    PreparedStatement prestate =
    SqlUtil.loadQuery(conn, "state.gxq", "STATES_update1",
param);

    // temporary parameter substitutions.
    prestate.setString(param.getValInt("STATE"), "CA");
    prestate.setInt(param.getValInt("NEW_POP"), 10000);
    prestate.execute();

     prestate.clearParameters();
    // temporary parameter substitutions.
    prestate.setString(param.getValInt("STATE"), "TX");
    prestate.setInt(param.getValInt("NEW_POP"), 10000);
    prestate.execute();
```

```
    conn.close();
}
catch (SQLException err)
{
}
```

Related Topics

com.iplanet.server.jdbc.iAS6.0RowSet class,
java.sql.Connection interface,
java.sql.PreparedStatement interface,
javax.sql.RowSet interface

loadQuery( )

Loads a query.

Syntax 1

Use any of the following three variants to initialize a RowSet, given a SELECT query.

```
static public boolean loadQuery(
   RowSet rs,
   String filepath,
   String queryname,
   IValList params)
```

Syntax 2

```
static public boolean loadQuery(
   RowSet rs,
   String query,
   IValList params)
```

Syntax 3

```
static public boolean loadQuery(
   RowSet rs,
   Reader r,
   String queryname,
   IValList params)
```

Syntax 4

Use any of the following three variants to generate a PreparedStatement, given an INSERT, UPDATE, or DELETE query.

```
static public PreparedStatement loadQuery(
   Connection conn,
   String filepath,
   String queryname,
   IValList params)
```

Syntax 5

```
static public PreparedStatement loadQuery(
   Connection conn,
   String query,
   IValList params)
```

Syntax 6

```
static public PreparedStatement loadQuery(
   Connection conn,
   Reader r,
   String queryname,
   IValList params)
```

rs. Specifies the rowset on which the query will act. The rowset handed in must have its Connection initialized; otherwise the loadQuery( ) call will fail.

filepath. The full pathname of the query file (.gxq file).

queryname. For Syntax 3 (using a RowSet), the name of the SELECT query in the query file. For Syntax 6 (using a Connection), the name of the INSERT, UPDATE, or DELETE query in the query file.

query. For Syntax 2 (using a RowSet), the SELECT query. For Syntax 5 (using a Connection), the INSERT, UPDATE, or DELETE query.

params. A list of parameter-value pairs that will be permanently replaced in the SELECT query.

r. A reader that points to the contents of a query file (.gxq file).

conn. Connection information needed to generate a PreparedStatement.

Usage

Use loadQuery( ) either to initialize a RowSet given a SELECT query, or to generate a PreparedStatement given an INSERT, UPDATE, or DELETE query. This method performs the following:

- removes comments of the form /* (slash-asterisk).
- replaces any parameters it finds in the SQL statement with values that match the passed-in parameter list.

- replaces parameters it finds in the SQL statement with ? (question mark) for values it cannot find in the parameter list.
- converts SQL statements to database-specific SQL for constructs such as left joins, right joins, or outer joins.

Return Value

The boolean syntax returns true if no errors were encountered or returns false for failure. The PreparedStatement syntax has the following return values: if no errors are encountered, loadQuery( ) returns the PreparedStatement object that references the query; otherwise, the method returns null for failure.

TemplateDataBasic class (*deprecated)*

TemplateDataBasic is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model, where similar functionality is provided through interfaces such as java.sql.ResultSet and javax.sql.RowSet.

For information about replacing TemplateDataBasic functionality in existing applications, see the *Migration Guide*.

The TemplateDataBasic class represents a memory-based, hierarchical source of data used for HTML template processing. It implements the ITemplateData Interface (deprecated), and provides methods for creating and managing this hierarchical data.

The most common sources of data used for template processing are result sets obtained from queries on supported relational database management systems. However, an AppLogic might need to obtain data from non-RDBMS sources. For example, an AppLogic might display a list of numbers generated from a formula, or it might display a list of processors available on the server machine and their CPU loads. To display such information, the AppLogic can create an instance of the TemplateDataBasic class, populate that instance with rows of hierarchical data, and then pass the TemplateDataBasic object to the Template Engine for processing by calling evalTemplate( ) or evalOutput( ) in the AppLogic class (deprecated).

Alternatively, to provide application-specific special processing and to hook into the template generation process, AppLogic can subclass the TemplateDataBasic class and override the member methods in the ITemplateData Interface (deprecated).

An AppLogic can create a flat or hierarchical data structure.

- For a flat data structure, create the data structure using TemplateDataBasic( ), then call rowAppend( ) for each row of data to be added, specifying the column name and data in each row.
- For a hierarchical data structure, proceed in the following sequence:
  a. Create the parent TemplateDataBasic( ) instance.
  b. Create the child TemplateDataBasic( ) instance.
  c. Add one or more rows to the child data structure using rowAppend( ) on the child instance.
  d. Define the start of a new parent row by using rowAppend( ) on the parent instance.
  e. Join the child data structure to the parent data structure using the parent's groupAppend( ).
  f. Repeat steps 2 through 5 for each subsequent group of data.

The number of nesting levels is limited only by system resources. One parent row can contain many joined child instances, in which case the AppLogic calls the parent's groupAppend( ) more than once after calling the parent's rowAppend( ).

To create a template data source, use the GX.CreateTemplateDataBasic( ) method, as shown in the following example:

```
TemplateDataBasic data = GX.CreateTemplateDataBasic("data");
```

Package

com.kivasoft.applogic

Methods

Method	Description
groupAppend( )	Links the specified child group to the current parent group.
rowAppend( )	Appends a new row of data to the current template data object or group.
TemplateDataBasic( )	Creates an empty template data object with the specified name.

Implements

ITemplateData Interface (deprecated)

Example 1

```
// Construct a flat result set
/* Create Data Object */
tdbSalesRev = new TemplateDataBasic("salesOffices");
/* Create records of data */
tdbSalesRev.rowAppend("office=New York;revenue=150M");
tdbSalesRev.rowAppend("office=Hong Kong;revenue=130M");
tdbSalesRev.rowAppend("office=Singapore;revenue=105M");
```

Example 2

```
// Construct a hierarchical result set with two child
// levels of data: Asia and Europe.

/* Create the hierarchical data object */
tdbContinents = new TemplateDataBasic("continents");

/* Create the Asia group */
tdbAsia = new TemplateDataBasic("countries");
/* Create child records for Asia group */
tdbAsia.rowAppend("country=China;currency=yuan");
tdbAsia.rowAppend("country=Japan;currency=yen");
tdbAsia.rowAppend("country=South Korea;currency=won");
tdbContinents.rowAppend("name=Asia");
/* Link child records to group */
tdbContinents.groupAppend(tdbAsia);

/* Create the Europe group in the same manner */
tdbEurope = new TemplateDataBasic("countries");
tdbEurope.rowAppend("country=France;currency=franc");
tdbEurope.rowAppend("country=Germany;currency=deutsche mark");
tdbEurope.rowAppend("country=Italy;currency=lire");
tdbContinents.rowAppend("name=Europe");
tdbContinents.groupAppend(tdbEurope);
```

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

ITemplateData Interface (deprecated)

groupAppend( )

Links the specified child group to the current parent group.

Syntax

```
public int groupAppend(
    TemplateDataBasic childName)
```

childName . Name of the child TemplateDataBasic object to link.

Usage

Use groupAppend( ) to define the hierarchical relationship from a parent row to child TemplateDataBasic objects.

Rules

- Call groupAppend( ) only after calling rowAppend( ). The child instance is associated with the last row from the last call to rowAppend( ) on the parent.
- The AppLogic must first create the parent and child objects using new TemplateDataBasic( ), then populate the child object with rows of data using rowAppend( ).

Tip

Use groupAppend( ) for hierarchical data objects only.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Construct a hierarchical result set with two child
// levels of data: Asia and Europe.

/* Create the hierarchical data object */
tdbContinents = new TemplateDataBasic("continents");

/* Create the Asia group */
tdbAsia = new TemplateDataBasic("countries");
/* Create child records for Asia group */
tdbAsia.rowAppend("country=China;currency=yuan");
tdbAsia.rowAppend("country=Japan;currency=yen");
tdbAsia.rowAppend("country=South Korea;currency=won");
tdbContinents.rowAppend("name=Asia");
/* Link child records to group */
tdbContinents.groupAppend(tdbAsia);

/* Create the Europe group in the same manner */
```

```
tdbEurope = new TemplateDataBasic("countries");
tdbEurope.rowAppend("country=France;currency=franc");
tdbEurope.rowAppend("country=Germany;currency=deutsche mark");
tdbEurope.rowAppend("country=Italy;currency=lire");
tdbContinents.rowAppend("name=Europe");
tdbContinents.groupAppend(tdbEurope);
```

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

ITemplateData Interface (deprecated)

rowAppend( )

Appends a new row of data to the current template data object or group.

Syntax

```
public int rowAppend(
   String rowString)
```

rowString . String containing a series of column name and value pairs, separated by semi-colons, using the following format:

```
"column1=value1[;column2=value2[...]]"
```

The columns must be identical for each rowAppend( ) call within the same TemplateDataBasic object.

Usage

Use rowAppend( ) to populate the template data object with rows of data.

Rule

AppLogic must first create the template data object using TemplateDataBasic( ).

Tip

Add rows in the sequence in which you want the Template Engine to process them. The template data object is processed in physical order only. AppLogic can only append rows to the data object. It cannot subsequently insert, delete, or sort records in the template data object.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Construct a hierarchical result set with two child
// levels of data: Asia and Europe.

/* Create the hierarchical data object */
tdbContinents = new TemplateDataBasic("continents");

/* Create the Asia group */
tdbAsia = new TemplateDataBasic("countries");
/* Create child records for Asia group */
tdbAsia.rowAppend("country=China;currency=yuan");
tdbAsia.rowAppend("country=Japan;currency=yen");
tdbAsia.rowAppend("country=South Korea;currency=won");
tdbContinents.rowAppend("name=Asia");
/* Link child records to group */
tdbContinents.groupAppend(tdbAsia);

/* Create the Europe group in the same manner */
tdbEurope = new TemplateDataBasic("countries");
tdbEurope.rowAppend("country=France;currency=franc");
tdbEurope.rowAppend("country=Germany;currency=deutsche mark");
tdbEurope.rowAppend("country=Italy;currency=lire");
tdbContinents.rowAppend("name=Europe");
tdbContinents.groupAppend(tdbEurope);
```

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

ITemplateData Interface (deprecated)

TemplateDataBasic( )

Creates an empty template data object with the specified name.

Syntax

```
public TemplateDataBasic(
   String dataObjectName)
```

dataObjectName . Name of the parent or child data object referred to in the template.

Usage

Use new TemplateDataBasic( ) to create parent and child data objects.

Rule

The specified data object name must be unique within this template data object.

Tips

- Use rowAppend( ) to populate this data object with rows of data.
- For hierarchical template data objects, use groupAppend( ) to define the hierarchy among TemplateDataBasic objects.
- The specified data object name must be unique within the hierarchical result set.
- Create parent and child groups in the sequence in which you want the Template Engine to process them. The template data object is processed in physical order only.

Return Value

An empty TemplateDataBasic object, or null for failure.

Example

```
// Construct a hierarchical result set with two child
// levels of data: Asia and Europe.

/* Create the hierarchical data object */
tdbContinents = new TemplateDataBasic("continents");

/* Create the Asia group */
tdbAsia = new TemplateDataBasic("countries");
/* Create child records for Asia group */
tdbAsia.rowAppend("country=China;currency=yuan");
tdbAsia.rowAppend("country=Japan;currency=yen");
tdbAsia.rowAppend("country=South Korea;currency=won");
tdbContinents.rowAppend("name=Asia");
/* Link child records to group */
tdbContinents.groupAppend(tdbAsia);

/* Create the Europe group in the same manner */
tdbEurope = new TemplateDataBasic("countries");
tdbEurope.rowAppend("country=France;currency=franc");
tdbEurope.rowAppend("country=Germany;currency=deutsche mark");
tdbEurope.rowAppend("country=Italy;currency=lire");
tdbContinents.rowAppend("name=Europe");
tdbContinents.groupAppend(tdbEurope);
```

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

ITemplateData Interface (deprecated)

TemplateMapBasic class *(deprecated)*

TemplateMapBasic is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model.

For information about replacing TemplateMapBasic functionality in existing applications, see the *Migration Guide*.

The TemplateMapBasic class represents an object that contains one or more mappings between fields in an HTML template and the data used to replace those fields. It provides a method for defining these mappings before processing the template using evalTemplate( ) or evalOutput( ) in the AppLogic class (deprecated).

Fields in the HTML template are defined using special GX markup tags. The data, which the Template Engine uses to replace those fields dynamically at runtime, can come from any of the following sources: a calculated value, a column in a result set, a field in an ITemplateData template data object, or a field from a map object.

Before calling evalTemplate( ) or evalOutput( ) in the AppLogic class, an AppLogic uses the put( ) or putString( ) method in the TemplateMapBasic class to link the field name in the GX markup tag with a precomputed value or a named column or field in the data source. After defining the mappings, the AppLogic passes the populated ITemplateMapBasic object as the map parameter in evalTemplate( ) or evalOutput( ). The Template Engine uses these mappings during template processing to dynamically transfer data values from the data source to the HTML output report.

Mapping allows the AppLogic to use the same template for multiple data sources with different column names, for a data source whose schema changes over time, or for memory-based data sources defined using a TemplateDataBasic object.

To create a template map, use the GXCreateTemplateMapBasic( ) method, as shown in the following example:

```
TemplateMapBasic map = GX.CreateTemplateMapBasic();
```

To provide application-specific special processing, the AppLogic can subclass TemplateMapBasic and override its getString( ) methodto hook into the Template Engine generation process. For example, AppLogic can intercept and filter data from a database before the Template Engine processes it.

Package

com.kivasoft.applogic

Methods

Method	Description
getString( )	Resolves the id attribute specified in a GX markup tag in the template being processed by the Template Engine. This method is called by the get( ) method in the ITemplateMap interface.
put( )	Maps the value assigned to the id attribute in the HTML template to another value. This method takes an IBuffer object (that contains a string expression) for the replacement value argument.
putString( )	Maps the value assigned to the id attribute in the HTML template to another value. This method takes a string argument for the replacement value.

Implements

ITemplateMap Interface (deprecated)

Example

```
// Create template map
TemplateMapBasic map = new TemplateMapBasic();
[... create a query ...]
// map columns in table to tags in template
map.putString("ParentsSelect", selStr);
map.putString("NAME", myName);
map.putString("CATEGORY", catStr);
// Evaluate the return template using the column mappings
if(evalTemplate(HTML, (ITemplateData) null, map)==0)
   return result("");
else
   return result("Failed to Generate HTML");
```

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated),
TemplateDataBasic class (deprecated),
ITemplateData Interface (deprecated),
ITemplateMap Interface (deprecated)

getString( )

Resolves the `id` attribute specified in a GX markup tag in the template being processed by the Template Engine. This method is called by the Template Engine.

Syntax

```
public String getString(
   String szExpr,
   IObject pData,
   IObject pMark)
```

szExpr. In the current GX markup tag in the HTML template being processed, the name of the field, or placeholder, assigned to the id attribute. Must be an identical match (case-sensitive).

pData. Specify null.

pMark. Specify null.

Usage

GX markup tags are used in an HTML template to identify where dynamic data appears in the output report. In the GX markup tags, the `id` attribute specifies any of the following items: the name of a flat query within a hierarchical query, a field in the hierarchical result set or TemplateDataBasic object, or an HTML template. The type of item specified in the `id` attribute depends on the `type` attribute that is specified in the same GX markup tag.

To provide application-specific special processing, an AppLogic can subclass the TemplateMapBasic class (deprecated) and override getString( ) to manipulate the Template Engine generation process. For example, an AppLogic can intercept and filter data from a database before the Template Engine processes it.

Rule

An AppLogic should use getString( ) only to override it after subclassing the TemplateMapBasic class (deprecated).

Return Value

A string that contains the value of the id mapping, or null for failure.

Example

The following code shows how getString( ) in a custom template map class is overridden to recalculate the current time each time it is encountered in a template:

```
import com.kivasoft.*;
class myTMB extends TemplateMapBasic {
   public String getString(String key, data, mark)
   {
      if (key.equals("CURTIME"))
         return new Date().toString();
      return super.getString(key, data, mark);
   }
}
```

put( )

Maps the value assigned to the id attribute in the HTML template to another value. This method takes an IBuffer object that contains a string expression for the replacement value.

Syntax

```
public int put(
   String szExpr,
   IBuffer pBuff)
```

szExpr . In the GX markup tag in the HTML template, the name of the field, or placeholder, assigned to the id attribute. Must be an identical match (case-sensitive).

pBuff. IBuffer object that contains the String expression to substitute for the specified template field name, such as:

- Calculated value, such as a number or date.
- Name of the column in the hierarchical result set that the Template Engine uses to process the template. In your template, the column name must begin with a “$” character.
- Name of a field in the TemplateDataBasic object that the Template Engine uses to process the template. In your template, the field name must begin with a “$” character.

Usage

Use put( ) to add template field / data source pairs to the template map before calling evalTemplate( ) or evalOutput( ) in the AppLogic class.

Tips

- To create an IBuffer object that contains the string expression to pass to put( ), use GX.CreateBufferFromString( ). Alternatively, use putString( ) to pass a string directly.
- The AppLogic can place the put( )method call inside a loop to construct the field map iteratively. For example, the AppLogic could use this technique to populate a map from a file, line by line.

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated),
TemplateDataBasic class (deprecated) ,
ITemplateData Interface (deprecated) ,
ITemplateMap Interface (deprecated)

putString( )

Maps the value assigned to the id attribute in the HTML template to another value. This method takes a string argument for the replacement value.

Syntax

```
public int putString(
   String szExpr,
   String szData)
```

szExpr. In the GX markup tag in the HTML template, the name of the field, or placeholder, assigned to the id attribute. Must be an identical match (case-sensitive).

szData. The String expression to substitute for the specified template field name, such as:

- Calculated value, such as a number or date. Use the string.valueOf( ) methods to convert calculated values to a string.
- Name of the column in the hierarchical result set that the Template Engine uses to process the template. In your template, the column name must begin with a “$” character.

- Name of a field in the TemplateDataBasic object that the Template Engine uses to process the template. In your template, the field name must begin with a “$” character.

Usage

Use putString( ) to add template field / data source pairs to the template map before calling evalTemplate( ) or evalOutput( ) in the AppLogic class.

Tip

The AppLogic can place the putString( ) method call inside a loop to construct the field map iteratively. For example, the AppLogic could use this technique to populate a map from a file, line by line.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Create template map
TemplateMapBasic map = new TemplateMapBasic();
[... create a query ...]
// map columns in table to tags in template
map.putString("ParentsSelect", selStr);
map.putString(ICatalogData.COL_NAME, myName);
map.putString(ICatalogData.COL_CATEGORYID, catStr);

// Evaluate the return template using the column mappings
if(evalTemplate(HTML, (ITemplateData) null, map)==0)
   return result("");
else
   return result("Failed to Generate HTML");
```

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

TemplateDataBasic class (deprecated)

ITemplateData Interface (deprecated)

ITemplateMap Interface (deprecated)

Util class *(deprecated)*

The Util class is deprecated and is provided for backward compatibility only. New applications developed using standard JDBC calls do not need the functionality provided by the Util class.

The Util class is a utility class that contains static methods for converting Java Date objects to iPlanet Application Server date/time formats and IBuffer values to string. Like the methods in the Math and System classes in the Java Class Library, the methods in the Util class can be used anywhere regardless of whether an instance of the class exists or not.

You call the methods in the Util class by using the following convention: Util.*method*. The following example shows how you call toString( ), a method in the Util class, to convert data in a returned IBuffer object to a string value:

```
String groupbyString;

groupbyString = Util.toString(qry.getGroupBy());
```

Package

com.kivasoft.util

Methods

Method	Description
dateToDate( )	Converts a Java Date object and returns a string containing the date value in iPlanet Application Server date format.
dateToTime( )	Converts a Java Date object and returns a string containing the time value in iPlanet Application Server time format.
dateToTimeStamp( )	Converts a Java Date object and returns a string containing the date and time value in iPlanet Application Server time stamp format.
toString( )	Converts data in an IBuffer object to a string value.

dateToDate( )

Converts a Java Date object and returns a string containing the date value in the iPlanet Application Server date format.

Syntax

```
public static String dateToDate(
   Date date)
```

date. Date object containing the date value to convert. The specified date must be a Java Date object.

Usage

Use dateToDate( ) when working with dates and operations in the Data Access Engine (DAE).

Tip

See the java.util.Date class for additional options for obtaining and formatting date and time values.

Return Value

String representing the date portion of the Date object, or null for failure.

Example 1

```
Date startDate=new Date(String.valueOf(month)+"/1/"+year);

Date endDate=new Date(String.valueOf(month)+"/"+numDays+"/"+year+"
23:59:59");

log(month+"/"+numDays+"/"+year);

log(startDate.toString()+" "+dateToDate(startDate));

log(endDate.toString()+" "+dateToDate(endDate));
```

Example 2

```
// Get and display date portion of current date as a string.

String todaysDate;

todaysDate=dateToDate(new date());

System.out.println("Today, in iPlanet Application Server format, is
" + todaysDate + ".");
```

Related Topics

dateToTime( )

dateToTimeStamp( )

dateToTime( )

Converts a Java Date object and returns a string containing the time value in the iPlanet Application Server time format.

Syntax

```
public static String dateToTime(
    Date date)
```

date. Date object containing the time value to convert. The specified date must be a Java Date object.

Usage

Use dateToTime( ) when working with time and operations in the Data Access Engine (DAE).

Return Value

String representing the time portion of the Date object, or null for failure.

Tip

See the java.util.Date class for additional options for obtaining and formatting date and time values.

Example 1

```
Date startDate=new Date(String.valueOf(month)+"/1/"+year);

Date endDate=new Date(String.valueOf(month)+"/"+numDays+"/"+year+"
23:59:59");

log(month+"/"+numDays+"/"+year);

log(startTime.toString()+" "+dateToTime(startDate));

log(endTime.toString()+" "+dateToTime(endDate));
```

Example 2

```
// Get and display time portion of current date as a string.

String timeRightNow;

timeRightNow=dateToTime(new date());

System.out.println("Time right now, in iPlanet Application Server
format, is " + todaysDate + ".");
```

Related Topics

dateToDate( )

dateToTimeStamp( )

dateToTimeStamp( )

Converts a Java Date object and returns a string containing the date and time value in the iPlanet Application Server time stamp format.

Syntax

```
public static String dateToTimeStamp(
   Date date)
```

date. Date object containing the value to convert. The specified date must be a Java Date object.

Usage

Use dateToTimeStamp( ) when working with dates and times, as well as operations in the Data Access Engine (DAE).

Tip

See the java.util.Date class for additional options for obtaining and formatting date and time values.

Return Value

String representing the date and time portions of a date.

Example 1

```
Date startDate=new Date(String.valueOf(month)+"/1/"+year);

Date endDate=new Date(String.valueOf(month)+"/"+numDays+"/"+year+"
23:59:59");

log(month+"/"+numDays+"/"+year);

log(startTime.toString()+" "+dateToTimeStamp(startDate));

log(endTime.toString()+" "+dateToTimeStamp(endDate));
```

Example 2

```
// Get and display time portion of current date as a string.

String dateTimeRightNow;

dateTimeRightNow=dateToTimeStamp(new date());

System.out.println("Date and time right now, in iPlanet Application
Server format, is " + dateTimeRightNow + ".");
```

Related Topics

dateToDate( )

dateToTime( )

toString( )

Converts data in an IBuffer object to a string value.

Syntax

```
public static String toString(
   IBuffer buffer)
```

buffer. IBuffer object containing the value to convert.

Usage

Use toString( ) to convert the IBuffer objects returned by the get**( ) methods in the IQuery Interface (deprecated) to string values.

Return Value

String representing the data in an IBuffer object.

Related Topics

IBuffer Interface (deprecated)

IQuery Interface (deprecated)

Chapter 3

Interfaces

This chapter provides reference material on the interfaces in the iPlanet Application Server Foundation Class Library.

The following interfaces are described in this chapter:

HttpServletRequest2 Interface

This interface extends the standard javax.servlet.http.HttpServletRequest interface.

Although anyone developing an application for iPlanet Application Server can use HttpServletRequest2, this interface is typically used in components generated by iplanet Application Builder. HttpServletRequest2 provides methods for handling servlet errors, as well as providing additional support for servlets.

One method in particular, the getAppLogic( ) method, is useful to developers outside the iAB environment. Through the getAppLogic( ) method, the HttpServletRequest2 interface supports servlet access to AppLogics, allowing applications to call methods on AppLogics.

Package

com.netscape.server.servlet.extension

Methods

Method	Description
convertITemplateDataToResultSet( )	Creates a ResultSet wrapper for an ITemplateData object.
dispatchAction( )	Calls a method corresponding to a form-action in a servlet.
formActionHandlerExists( )	Determines whether the servlet has form-actions defined on it.
getAppLogic( )	Returns a handle to the AppLogic instance (ServletRunner) that served this request.
getDefaultTemplate( )	Retrieves the name of the file that handles output.
getErrorCodes( )	Retrieves a vector of error codes.
getErrorMsgs( )	Retrieves a vector of error messages.
getErrorVars( )	Retrieves a vector of error variables.
getHttpSessionBean( )	Retrieves an instance of a session object that is defined by the programmer.
getServletErrorHandler( )	Retrieves the object that handles validation errors.
setDefaultTemplate( )	Sets the default template to another file.

Method	Description
setServletErrorHandler( )	Sets the IServletErrorHandler that is used to process errors.
validate( )	Validates input parameters and session variables.

Related Topics

IServletErrorHandler Interface

convertITemplateDataToResultSet( )

Creates a ResultSet wrapper for an ITemplateData object.

Syntax

```
public static ResultSet convertITemplateDataToResultSet(
   String groupName,
   ITemplateData templateData)
```

groupName. The group name of the specified ITemplateData.

templateData. The ITemplateData to be wrapped.

Usage

The convertITemplateDataToResultSet( ) method takes an ITemplateData object and wraps it so that it conforms to the specifications of a JDBC ResultSet object.

Use this method only if you have HTML templates that were developed for a previous version of iAS, and you want to run them in the JSP engine.

Rules

Hierarchical ITemplateData objects are not supported, so do not specify one as a parameter.

Return Value

A ResultSet object.

dispatchAction( )

Calls a method corresponding to a form-action in a servlet.

Syntax

```
public abstract int dispatchAction(
   HttpServletResponse response,
   HttpServlet servlet) throws ServletException, IOException
```

response. The HttpServletResponse object.

servlet. The HttpServlet object on which the actionHandler will be invoked.

Usage

Use this method to handle form actions. For example, if an application user presses a button in a web page form, the dispatchAction( ) method calls the button handler method that corresponds to the specific form and action.

The form, action, and corresponding handler are defined by the servlet developer in the servlet's NTV file.

Tip

It is useful to call formActionHandlerExists( ) before dispatchAction( ) to determine whether the servlet has form-actions defined on it.

Return Value

Returns NO_ERROR (a value of 0) if the method succeeds; otherwise, returns ERROR_NO_DISPATCH_HANDLER. (a value of 1), which means that dispatching cannot be done. If an event handler exists and is called, then dispatchAction( ) returns that handler's return value.

Related Topics

formActionHandlerExists( )

formActionHandlerExists( )

Determines whether the servlet has form-actions defined on it.

Syntax

```
public abstract boolean formActionHandlerExists()
```

Usage

This method accesses the NTV list and, by doing so, determines whether the servlet has form-actions defined on it. This information is useful to the dispatchAction( ) method, so you typically call formActionHandlerExists( ) before dispatchAction( ).

Tip

This method appears in code generated by iPlanet Application Builder. If you are not using iAB, you typically will not need to call this method.

Return Value
Returns true if form-actions are defined on the servlet.

Related Topics
dispatchAction( )

getAppLogic( )
Returns a handle to the AppLogic instance (ServletRunner) that served this request.

Syntax
```
public abstract AppLogic getAppLogic()
```

Usage
Use this method when you want to access the ServletRunner AppLogic. The getAppLogic( ) method lets developers call any AppLogic methods.

Tip
Deprecated AppLogic methods should not be called. For example, calling methods that stream data, such as streamResult( ), streamResultBinary( ), and streamResultHeader( ), may conflict with similar methods in HTTPResponse that also stream data.

Return Value
An AppLogic object.

Related Topics
AppLogic class

getDefaultTemplate( )
Retrieves the name of the file that handles output.

Syntax
```
public abstract String getDefaultTemplate()
```

Usage
In some cases, a servlet's NTV file indicates a default servlet or JSP file that handles output. If so, calling getDefaultTemplate( ) retrieves this NTV setting.

Tip
This method appears in code generated by iPlanet Application Builder. If you are not using iAB, you typically will not need to call this method.

Return Value
A String value representing the name of the default template.

Related Topics
setDefaultTemplate( )

getErrorCodes( )
Retrieves a vector of error codes.

Syntax
```
public abstract Vector getErrorCodes()
```

Return Value
A vector of error codes that correspond to the input variables that failed validation. A code is associated with each type of data validated, as summarized in the following table:

Data to Validate	Validation Rule	Description
Number	VALIDATE_NUMBER	Data is numerical.
Integer	VALIDATE_INTEGER	Data is an integer.
Positive integer	VALIDATE_POSITIVE_INTEGER	Data is a positive integer.
Alphabetic	VALIDATE_ALPHABETIC	Data is alphabetic, a-z and/or A-Z.
US phone number	VALIDATE_US_PHONE	Data consists only of 10 digits and optionally the characters (,), and -.
International phone number	VALIDATE_INTL_PHONE	Data consists only of numbers and the characters (,), +, and -.
Email	VALIDATE_EMAIL	Data is in the format *a*@*b*.*c*
Social Security Number	VALIDATE_SSN	Data consists only of 9 digits and optionally the character -.
Date	VALIDATE_DATE	Data is in the format "MM-DD-YY" or "MM-DD-YYYY". Month, day, and year are validated accordingly.
Day	VALIDATE_DAY	Data is an integer between 1 and 31 inclusive.
Month	VALIDATE_MONTH	Data is an integer between 1 and 12 inclusive.

Data to Validate	Validation Rule	Description
Year	VALIDATE_YEAR	Data is an integer between 1 and 99 inclusive, or between 1000 and 9999 inclusive.
US zipcode	VALIDATE_US_ZIPCODE	Data consists of only 5 or 9 digits and optionally the character -.

getErrorMsgs( )

Retrieves a vector of error messages.

Syntax

```
public abstract Vector getErrorMsgs()
```

Return Value

A vector of error messages that correspond to the input variables that failed validation. A message is associated with each type of data validated, as summarized in the following table:

Validation Rule	Error Message
VALIDATE_NUMBER	Wrong number format!
VALIDATE_INTEGER	Wrong integer format!
VALIDATE_POSITIVE_INTEGER	Wrong positive integer format!
VALIDATE_ALPHABETIC	Wrong alphabetic format!
VALIDATE_US_PHONE	Wrong US phone format!
VALIDATE_INTL_PHONE	Wrong international phone format!
VALIDATE_EMAIL	Wrong email format!
VALIDATE_SSN	Wrong social security format!
VALIDATE_DATE	Wrong date format!
VALIDATE_DAY	Wrong day format!
VALIDATE_MONTH	Wrong month format!
VALIDATE_YEAR	Wrong year format!
VALIDATE_US_ZIPCODE	Wrong zip code format!

getErrorVars( )

Retrieves a vector of input variables.

Syntax

```
public abstract Vector getErrorVars()
```

Return Value

A vector of the names of input variables that failed to validate.

getHttpSessionBean( )

Retrieves an instance of a session object that is defined by the programmer.

Syntax 1

Use this syntax in most situations.

```
public abstract HttpSession getHttpSessionBean()
```

Syntax 2

Use this syntax to explicitly indicate whether to create the session or not.

```
public abstract HttpSession getHttpSessionBean(
   boolean create)
```

create. Supply a value of true if the session is to be created; otherwise, specify false.

Usage

The getHttpSessionBean( ) method retrieves the HttpSession or HttpSession2 object that is an instance of the class designated by the httpSessionBeanClass setting. This setting is defined in the SessionInfo section of an application's appInfo.ntv file. This setting allows application developers to provide bean-like HttpSession implementations.

Tip

This method appears in code generated by iPlanet Application Builder. If you are not using iAB, you typically will not need to call this method.

Return Value

An HttpSession or HttpSession2 object representing an instance of the class designated by httpSessionBeanClass.

Related Topics

getSession( ) in the javax.servlet.http.HttpServletRequest interface

getServletErrorHandler( )

Retrieves the object that handles validation errors.

Syntax

```
public abstract IServletErrorHandler getServletErrorHandler()
```

Usage

This method is used to retrieve the IServletErrorHandler object that handles errors that occur during validation. You can retrieve the object elsewhere if you need to call the included functionality.

Tip

This method appears in code generated by iPlanet Application Builder. If you are not using iAB, you typically will not need to call this method.

Return Value

The IServletErrorHandler object.

Related Topics

IServletErrorHandler interface

setDefaultTemplate( )

Sets the default template to another file.

Syntax

```
public abstract void setDefaultTemplate(
   String template)
```

template. The new name of the default template; the name applies only for this invocation of the servlet.

Usage

The default template is defined in a servlet's NTV file. However, in some cases it's useful to reset the default template to another file. To do so, call setDefaultTemplate( ). To retrieve the name of the current default template, call getDefaultTemplate( ).

Tip

This method appears in code generated by iPlanet Application Builder. If you are not using iAB6.0, you typically will not need to call this method.

Related Topics

getDefaultTemplate( )

setServletErrorHandler( )

Sets the IServletErrorHandler that is used to process errors.

Syntax

```
public abstract void setServletErrorHandler(
   IServletErrorHandler handler)
```

handler. The error handler to be set.

Usage

A servlet error handler is always set on a request object by default. However, you may want to implement your own, so the setServletErrorHandler( ) method is provided to allow that.

This method sets the IServletErrorHandler that is used to process errors that happen during validation (and possibly elsewhere). To retrieve the IServletErrorHandler, call getServletErrorHandler( ).

Tip

This method appears in code generated by iPlanet Application Builder. If you are not using iAB, you typically will not need to call this method.

Related Topics

getServletErrorHandler( )

validate( )

Validates input parameters and session variables.

Syntax 1

Use this version to validate all input parameters and session variables listed in the servlet's NTV file. This version also streams out an error on failure.

```
public abstract boolean validate(
   HttpServletResponse response)
   throws ServletException, IOException
```

Syntax 2

Use this version to validate specific input parameters and session variables (or those listed in the servlet's NTV file). This version also streams out an error on failure.

```
public abstract boolean validate(
   String params[],
   String sessionVars[],
   HttpServletResponse response)
   throws ServletException, IOException
```

Syntax 3

Use this version to validate specific input parameters and session variables (or those listed in the servlet's NTV file). No error streaming occurs.

```
public abstract boolean validate(
   String params[],
   String sessionVars[])
```

Syntax 4

Use this version to validate all input parameters and session variables listed in the servlet's NTV file. No error streaming occurs.

```
public abstract boolean validate()
```

response. The HttpServletResponse object.

params. The array of parameters to validate. Specify NULL to validate all input parameters listed in the NTV file.

sessionVars. The array of session variables to validate. Specify NULL to validate all session variables listed in the NTV file.

Usage

Use this method whenever you want to validate parameters. There are four ways to invoke the validate( ) method. One version validates all input and session variables listed in a servlet's NTV file. Another version validates a specified array of input parameters or session variables. For each of these two versions, you can either ignore failure conditions or stream an error on failure.

When validate( ) is called and a validation error is detected, validate( ) then calls either of two IServletErrorHandler methods: handleInputValueError( ) or handleSessionVariableError( ). The previous two methods might return a positive number, meaning that a real error occurred and is queued into the error-specific vectors. If this happens, then streamError( ) is called once before validate( ) returns (assuming the validate( ) call is one of the two variants that streams errors).

The validate( ) method passes in one of the following error types as input to the two error handler methods, handleInputValueError( ) or handleSessionVariableError( ):

Error Type	Value	Error because data must be ...
ERROR_NUMBER	5	a number.
ERROR_INTEGER	6	an integer.
ERROR_POSITIVE_INTEGER	7	a positive integer.
ERROR_ALPHABETIC	8	alphabetic characters (a-z, A-Z, or both).
ERROR_US_PHONE	9	a phone number containing at most 10 digits and optional characters (,), and -.
ERROR_INTL_PHONE	10	a phone number containing only digits and optional characters, (,), +, and -.
ERROR_EMAIL	11	an address of the form a@b.c.
ERROR_SSN	12	a Social Security Number containing 9 digits and an optional - character.
ERROR_DATE	13	a date of the form MM-DD-YY or MM-DD-YYYY.
ERROR_DAY	14	an integer between 1 and 31 inclusive.
ERROR_MONTH	15	an integer between 1 and 12 inclusive.
ERROR_YEAR	16	an integer between 1 and 99 inclusive or between 1000 and 9999 inclusive.
ERROR_ZIP	17	a ZIP code containing either 5 digits or 9 digits, and an optional - character.

Return Value

Returns true if the validation was successful; otherwise, returns false.

Related Topics

IServletErrorHandler interface

HttpSession2 Interface

The HttpSession2 interface is used within JSPs and servlets to share session state with AppLogics.

iPlanet Application Server already supports javax.servlet.http.HttpSession as a standard interface to iPlanet Application Server sessions. But the application server also provides HttpSession2, a iAS-specific interface. HttpSession2 gives servlets direct access to iPlanet Application Server sessions. Therefore, servlet programmers can use HttpSession2 to share sessions between AppLogics and servlets.

Sharing sessions is useful when you want to migrate an application from iPlanet Application Server 2.x to iPlanet Application Server 6.0. When migrating an application, one of the tasks is to rewrite AppLogics into servlets. Furthermore, when you rewrite AppLogics, you typically need to use iAS-specific servlet interfaces, such as HttpSession2 and HttpServletRequest2. The resulting servlet is nonstandard; however, the alternate approach—mixing AppLogics and standard servlets—is not recommended.

In servlets, a session is an instance of HttpSession. But in AppLogics, session data is an IValList object. An AppLogic stores integers, strings, and blobs (byte arrays) in a session, whereas a servlet stores serializable objects in a session. As a result, there is no immediate mapping between what an AppLogic stores and and what a servlet stores in a session (except for strings).

The HttpSession2 interface solves the issue of sharing session data. HttpSession2 provides methods for storing and retrieving integers, strings, blobs, and user login data—methods that parallel what an AppLogic developer uses. In this way, HttpSession2 enables sessions to work back and forth across AppLogics and servlets.

HttpSession2 provides loginSession( ) and logoutSession( ) for servlets to share the AppLogic session API. These two methods are typically used with isAuthorized( ), as is done for AppLogics. Servlets are also registered with an access control list, so that a secure session established in an AppLogic can be used in a servlet, and vice versa.

Package

com.netscape.server.servlet.extension

Methods

Method	Description
getBytes( )	Returns the byte array defined by the specified name in the session.
getInt( )	Returns the integer defined by the specified name in the session.
getString( )	Returns the string defined by the specified name in the session.

Method	Description
isAuthorized( )	Checks whether the current user has a specified permission.
loginSession( )	Logs an authorized user into a session with a secured application.
logoutSession( )	Removes a user's association with a session.
putBytes( )	Defines a name in the session to have a specified byte array value.
putInt( )	Defines a name in the session to have a specified integer value.
putString( )	Defines a name in the session to have a specified String value.

getBytes( )

Returns the byte array defined by the specified name in the session.

Syntax

```
public abstract byte[] getBytes(
   String name)
```

name. The name whose value is to be returned.

Return Value

The byte array defined by name in the session, or null if name is not defined.

Related Topics

putBytes( )

getInt( )

Returns the integer defined by the specified name in the session.

Syntax

```
public abstract int getInt(
   String name)
```

name. The name whose value is to be returned.

Return Value

The integer defined by name in the session, or -1 if name is not defined.

Related Topics

putInt( )

getString( )

Returns the string defined by the specified name in the session.

Syntax

```
public abstract String getString(
   String name)
```

name. The name whose value is to be returned.

Return Value

The string defined by name in the session, or null if name is not defined.

Related Topics

putString( )

isAuthorized( )

Checks whether the current user has a specified permission.

Syntax

```
public abstract boolean isAuthorized(
   String acl,
   String permission)
```

acl. The access control list in which to check for the permission.

permission. The permission to check for.

Usage

Use isAuthorized( ) in portions of the code where application security is enforced through Access Control Lists (ACLs). This method lets an application check a specified ACL to determine whether a user has permission to execute a servlet (or AppLogic) or to perform a particular action. The application can use the result of isAuthorized( ) as a condition in an If statement. It can, for example, return a message to users who are denied access to a servlet (or AppLogic).

Each ACL is defined in the registry and maps users to privileges such as read and write. Application developers should obtain the list of registered ACLs, users and groups from the server administrator who created these items. ACLs are created through the Enterprise Administrator tool or through the kreg tool.

Rule

Before calling isAuthorized( ), the application must create a session. The user must also be logged in with loginSession( ).

Return Value
Returns true if the authorization check succeeds; otherwise, returns false.

Related Topics
loginSession( )

loginSession( )
Logs an authorized user into a session with a secured application.

Syntax
```
public abstract boolean loginSession(
   String user,
   String password)
```

user. The login user name.

password. The user password.

Usage
The loginSession( ) method logs the named user in the session, using the given password. Logging in associates the user with the session so that the application can control authorization of AppLogic and servlet access.

Call loginSession( ) after creating or retrieving a user session. loginSession( ) checks the passed in login name and password against the user names and passwords stored in the iPlanet Application Server (the administrator sets up and manages this information) and logs the user into the session if the login name and password are valid.

If login is successful, a security credential object is created and associated with the session. The server checks this security credential object each time it receives an AppLogic or servlet request, and verifies if the user has execute permission for the AppLogic or servlet.

Using loginSession( ) in conjunction with isAuthorized( ), an application can ensure that only authorized users can take certain actions, such as executing AppLogics or servlets.

Tip
The server administrator creates users and passwords and manages access to AppLogics, servlets, and specified resources, such as sales or forecast reports. During the development and debugging phases, application developers can use the ldapmodify tool to create users, groups, and ACLs in the LDIF file. These tasks cannot be done programmatically.

Return Value
Returns true if the login is successful.

Related Topics
isAuthorized( ), logoutSession( )

logoutSession( )
Removes a user's association with a session.

Syntax
```
public abstract int logoutSession()
```

Usage
AppLogics or servlets call loginSession( ) to log into a session with a secured application. If loginSession( ) was called, you must call logoutSession( ) when the user exits the application or the secured portion of it.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
isAuthorized( ), loginSession( )

putBytes( )
Defines a name in the session to have a specified byte array value.

Syntax
```
public abstract void putBytes(
   String name,
   byte[] value)
```

name. The name to be defined.

value. The byte array value to assign to name.

Related Topics
getBytes( )

putInt( )
Defines a name in the session to have a specified integer value.

Syntax

```
public abstract void putInt(
    String name,
    int value)
```

name. The name to be defined.

value. The integer value to assign to name.

Related Topics

getInt( )

putString( )

Defines a name in the session to have a specified String value.

Syntax

```
public abstract void putString(
    String name,
    String value)
```

name. The name to be defined.

value. The String value to assign to name.

Related Topics

getString( )

IAppEvent Interface *(deprecated)*

IAppEvent is deprecated and is provided for backward compatibility only. New applications should use the iPlanet Application Server API's two replacement interfaces: IAppEventMgr and IAppEventObj.

The IAppEvent interface represents the defined events an application supports. An AppLogic can define events that are triggered at a specified time or times or when triggered explicitly.

Currently, an AppLogic can execute two actions when an event is triggered:

- Run a specified AppLogic
- Send an email

Events are stored persistently in the iPlanet Application Server, and are removed only when your application explicitly deletes them. They are typically used to schedule routine administrative tasks, such as making back-ups or getting statistics.

The IAppEvent interface defines methods for registering, triggering, enabling, disabling and deleting events. To create an instance of the IAppEvent interface, use the getAppEvent( ) method in the AppLogic class.

Package

com.kivasoft

Methods

Method	Description
deleteEvent( )	Removes a registered event from iPlanet Application Server.
disableEvent( )	Disables a registered event.
enableEvent( )	Enables a registered event.
enumEvents( )	Enumerates through the list of registered events.
queryEvent( )	Returns the properties of a registered event.
registerEvent( )	Registers a named event for use in applications.
setEvent( )	Triggers a registered event.

Example

The following example shows AppLogic code that registers two application events:

- The first event runs the RepGenAgent AppLogic at 5 AM everyday
- The second event emails a report generated by RepGenAgent at 6 AM everyday

```
package GXApp.appevent;
import java.lang.*;
import java.util.*;
import java.io.*;
import com.kivasoft.*;
import com.kivasoft.applogic.*;
```

```
import com.kivasoft.util.*;
import com.kivasoft.types.*;
import com.kivasoft.appevent.*;

public class ReportAgent extends AppLogic
{
   static final java.lang.String eventName1 = "RepGenEvent";
   static final java.lang.String eventName2 = "ReportEvent";

   public int execute()
   {
      // Create IValLists to pass information
      // for appevent registration of the events
      IValList eventOutput;
      IValList eventInput1 = GX.CreateValList();
      IValList eventInput2 = GX.CreateValList();

      if ((eventInput1 == null) || (eventInput2 == null))
         return streamResult("Cannot create ValList<br>");

      // Get the appevent manager
      IAppEvent appEvent = getAppEvent();

      if (appEvent == null)
         return streamResult("Cannot get AppEvent<br>");

      // Add the RepGenAgent appevent name to the vallist

eventInput1.setValString(GX_AE_RE_KEY_NAME.GX_AE_RE_KEY_NAME,
     eventName1);

      // Set the appevent state to be enabled

eventInput1.setValInt(GX_AE_RE_KEY_STATE.GX_AE_RE_KEY_STATE,
     GX_AE_RE_ES_FLAG.GX_AE_RE_EVENT_ENABLED);

      // Set the appevent time to be 05:00:00 hrs everyday

eventInput1.setValString(GX_AE_RE_KEY_TIME.GX_AE_RE_KEY_TIME,
     "5:0:0 */*/*");

      // Set the appevent action to run
      // the RepGenAgent applogic
   eventInput1.setValString(GX_AE_RE_KEY_NREQ.GX_AE_RE_KEY_NREQ,
    "GUIDGX-{620CB09B-1A1D-1315-AD23-0800207B918B}");

      // Register the event
      if (appEvent.registerEvent(eventName1, eventInput1) !=
        GXE.SUCCESS)
         return streamResult("Cannot register RepGenEvent<br>");

      // Add the ReportAgent appevent name to the vallist

eventInput2.setValString(GX_AE_RE_KEY_NAME.GX_AE_RE_KEY_NAME,
     eventName2);
```

```
      // Set the appevent state to be enabled

eventInput2.setValInt(GX_AE_RE_KEY_STATE.GX_AE_RE_KEY_STATE,
     GX_AE_RE_ES_FLAG.GX_AE_RE_EVENT_ENABLED);

      // Set the appevent time to be 06:00:00 hrs everyday

eventInput2.setValString(GX_AE_RE_KEY_TIME.GX_AE_RE_KEY_TIME,
     "6:0:0 */*/*");

      // Set the appevent action to send e-mail

eventInput2.setValString(GX_AE_RE_KEY_MTO.GX_AE_RE_KEY_MTO,
    "report@acme.com");

      // The content of the e-mail is in /tmp/report-file
      eventInput2.setValString(GX_AE_RE_KEY_MFILE.GX_AE_RE_KEY_
        MFILE, "/tmp/report-file");

      // The e-mail host running the SMTP server is mailsvr
      eventInput2.setValString(GX_AE_RE_KEY_MHOST.GX_AE_RE_KEY_
        MHOST, "mailsvr.acme.com");

      // The sender's e-mail address is admin@acme.com
      eventInput2.setValString(GX_AE_RE_KEY_SADDR.GX_AE_RE_KEY_
        SADDR, "admin@acme.com");

      // Register the event
      if (appEvent.registerEvent(eventName2, eventInput2) !=
        GXE.SUCCESS)
          return streamResult("Can not register ReportEvent<br>");
           return streamResult("Successfully Registered
RepGenEvent and
            ReportEvent<br>");
   }
}
```

Related Topics

getAppEvent( ) method in the AppLogic class

IAppEventMgr Interface

IAppEvent Interface (deprecated)

IValList interface

deleteEvent( )

Removes a registered event from the iPlanet Application Server.

Syntax

```
public int deleteEvent(
   String pEventName)
```

pEventName. The name of the registered event to delete.

Usage

Use deleteEvent( ) to remove an event that is no longer required. To temporarily stop a event from being triggered, use disableEvent( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

disableEvent( )

registerEvent( )

disableEvent( )

Disables a registered event.

Syntax

```
public int disableEvent(
   String pEventName)
```

pEventName. The name of the registered event to disable.

Usage

Use disableEvent( ) to temporarily stop an event from being triggered. The event is disabled until it is enabled with enableEvent( ). To remove an event from the iPlanet Application Server permanently, use deleteEvent( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

deleteEvent( )

enableEvent( )

registerEvent( )

enableEvent( )

Enables a registered event.

Syntax

```
public int enableEvent(
   String pEventName)
```

pEventName. The name of the registered event to enable.

Usage

Use enableEvent( ) to prepare a specified event for activation. Call enableEvent( ) after you register an event with registerEvent( ), or to enable a trigger that was disabled with disableEvent( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

disableEvent( )

registerEvent( )

enumEvents( )

Returns the list of registered events.

Syntax

```
public IEnumObject enumEvent()
```

Usage

Use enumEvents( ) to get information on all the registered events. The IEnumObject object returned by enumEvents( ) contains a set of IValList objects, one per event. Each IValList contains the properties assigned to the event when it was registered with registerEvent( ).

Tip

Use the methods in the IEnumObject interface to iterate through the contents of the returned IEnumObject object.

Example

The following AppLogic code shows how to use enumEvents( ) to get information on all the registered events and save it to a report:

```
// Open /tmp/report-file for writing the report
FileOutputStream outFile = null;
try {
   outFile = new FileOutputStream("/tmp/report-file");
} catch (IOException e) {
}
if (outFile == null)
   return streamResult("Can not open /tmp/report-file<br>");

ObjectOutputStream p = null;
try {
   p = new ObjectOutputStream(outFile);
} catch (IOException e) {
}
if (p == null)
    return streamResult("Cannot create ObjectOutputStream<br>");

// get appevent manager
IAppEvent appEvent = getAppEvent();

// Get the Enumeration object for all registered appevents
   IEnumObject enumObj = appEvent.enumEvents();

// Retrieve the count of registered appevents
int count = enumObj.enumCount();
try {
   p.writeObject("Number of Registered Events: ");
   p.writeInt(count);
} catch (IOException e) {
   return streamResult("Failed to write to report file<br>");
}

enumObj.enumReset(0);

while (count > 0) {
   IObject vListObj = enumObj.enumNext();
   if (vListObj instanceof IValList) {
      IValList vList = (IValList)vListObj;

         String name =

vList.getValString(GX_AE_RE_KEY_NAME.GX_AE_RE_KEY_NAME);
         try {
            p.writeObject("\nDefinitions for AppEvent: ");
            p.writeObject(name);
            p.writeObject("\n");
         } catch (IOException e) {
               return streamResult("Failed to write to report
file<br>");
```

```
            }

           // Reset the next GXVAL to retrieve
           // from ValList to be the first one
           vList.resetPosition();

            // Iterate through all the GXVALs in the vallist
           // and print them
```

Return Value

IEnumObject that contains the list of events, or null for failure.

Related Topics

getAppEvent( ) method in the AppLogic class

IValList interface

queryEvent( )

Returns the properties of a registered event.

Syntax

```
public IValList queryEvent(
   String pEventName)
```

pEventName. The name of the registered event to enable.

Usage

When an AppLogic calls registerEvent( ), it can specify any of the following:

- The initial state—enable or disabled—of the event
- The time the event is to be triggered
- The AppLogic to execute when the event is triggered
- The email to send when the event is triggered

Use queryEvent( ) to get the properties that were specified for a specific event.

Return Value

IValList object that contains information about the event.

Related Topics

registerEvent( )

registerEvent( )

Registers a named event for use in applications.

Syntax

```
public int registerEvent(
   String pEventName,
   IValList pValList)
```

pEventName. The name of the event to register.

pValList. The IValList object that specifies the properties of the event. The following table lists the keys and values you can specify:

Key	Value
GX_AE_RE_KEY_NAME. GX_AE_RE_KEY_NAME	A string representing the name of the event. If specified, the name must be the same one specified as the pEventName parameter.
GX_AE_RE_KEY_STATE. GX_AE_RE_KEY_STATE	A variable that specifies the initial state of the event: GX_AE_RE_ES_FLAG.GX_AE_RE_EVENT_ DISABLED GX_AE_RE_ES_FLAG.GX_AE_RE_EVENT_ ENABLED

Key	Value
GX_AE_RE_KEY_TIME. GX_AE_RE_KEY_TIME	The time at which the event will be triggered. Use the following format: hh:mm:ss W / DD / MM hh: 0 -23 mm: 0 - 59 ss: 0 - 59 W (day of the week): 0 - 6 with 0 = Sunday. DD (day of the month): 1 - 31 MM (month): 1 - 12 Each of these fields may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign indicating an inclusive range. For example, 2, 5 - 7:0:0 5 / * / * means the event is triggered at 2 AM, 5AM, 6 AM and 7 AM every Friday. The specification of days can be made by two fields: day of the month (DD) and day of the week (W). If both are specified, both take effect. For example, 1:0:0 1 / 15 / * means the event is triggered at 1 AM every Monday, as well as on the fifteenth of each month. To specify days by only one field, set the other field to *.
GX_AE_RE_KEY_NREQ. GX_AE_RE_KEY_NREQ	The AppLogic to execute when the event is triggered. Use the following format: GUIDGX-{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX?Param1=ABC&Param2=123
GX_AE_RE_KEY_MFILE. GX_AE_RE_KEY_MFILE*	The name of the file that contains the body of an email message.
GX_AE_RE_KEY_MTO. GX_AE_RE_KEY_MTO*	A comma separated list of users to send the email to.
GX_AE_RE_KEY_MHOST. GX_AE_RE_KEY_MHOST*	The name of the SMTP mail server.
GX_AE_RE_KEY_SADDR. GX_AE_RE_KEY_SADDR*	The sender’s email address.

* You must specify all of these items if sending email when the event is triggered.

Usage

Use registerEvent( ) to define each event your application will use. You can specify that a triggered event sends an email, or runs an AppLogic, or both.

Return Value

GXE.SUCCESS if the method succeeds.

Example

The following example shows how to define and register an application event:

```
static final java.lang.String eventName1 = "RepGenEvent";

// Get the appevent manager
IAppEvent appEvent = getAppEvent();

// Create IValList to pass information for
// registration of an event
IValList eventInput1 = GX.CreateValList();

// Add the RepGenAgent appevent name to the vallist

eventInput1.setValString(GX_AE_RE_KEY_NAME.GX_AE_RE_KEY_NAME,
eventName1);

// Set the appevent state to be enabled
   eventInput1.setValInt(GX_AE_RE_KEY_STATE.GX_AE_RE_KEY_STATE,
GX_AE_RE_ES_FLAG.GX_AE_RE_EVENT_ENABLED);

// Set the appevent time to be 05:00:00 hrs everyday

eventInput1.setValString(GX_AE_RE_KEY_TIME.GX_AE_RE_KEY_TIME,
"5:0:0 */*/*");

// Set the appevent action to run an AppLogic
   eventInput1.setValString(GX_AE_RE_KEY_NREQ.GX_AE_RE_KEY_NREQ,
"GUIDGX-{620CB09B-1A1D-1315-AD23-0800207B918B}");

// Register the event
if (appEvent.registerEvent(eventName1, eventInput1) !=
GXE.SUCCESS)
   return streamResult("Cannot register RepGenEvent<br>");
```

Related Topics

enableEvent( )

setEvent( )

getAppEvent( ) method in the AppLogic class

IValList interface

setEvent( )

Triggers a registered event.

Syntax

```
public int setEvent(
   String pEventName,
   int dwOverrideFlag,
   IValList pValList)
```

pEventName. The name of the event to trigger.

dwOverrideFlag. Specify 0 (zero) to trigger the event with the previously specified actions. To override the defined actions, you can specify the following:

- GX_AE_SE_OR_FLAG.GX_AE_SE_ACTION_NOMAIL if you don't want to send email when the event is triggered.
- GX_AE_SE_OR_FLAG.GX_AE_SE_ACTION_NOREQ if you don't want to run an AppLogic when the event is triggered.

pValList. The IValList object that specifies the event properties you want to override. Specify null to use the properties already defined for the event. The following table lists the keys and values you can specify:

Key	Value
GX_AE_RE_KEY_NAME. GX_AE_RE_KEY_NAME	A string representing the name of the event. If specified, the name must be the same one specified as the pEventName parameter.
GX_AE_RE_KEY_STATE. GX_AE_RE_KEY_STATE	A variable that specifies the initial state of the event: GX_AE_RE_ES_FLAG.GX_AE_RE_EVENT_DISABLED GX_AE_RE_ES_FLAG.GX_AE_RE_EVENT_ENABLED

Key	Value
GX_AE_RE_KEY_TIME. GX_AE_RE_KEY_TIME	The time at which the event will be triggered. Use the following format: hh:mm:ss W / DD / MM hh: 0 -23 mm: 0 - 59 ss: 0 - 59 W (day of the week): 0 - 6 with 0 = Sunday. DD (day of the month): 1 - 31 MM (month): 1 - 12 Each of these fields may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign indicating an inclusive range. For example, 2, 5 - 7:0:0 5 / * / * means the event is triggered at 2 AM, 5AM, 6 AM and 7 AM every Friday. The specification of days can be made by two fields: day of the month (DD) and day of the week (W). If both are specified, both take effect. For example, 1:0:0 1 / 15 / * means the event is triggered at 1 AM every Monday, as well as on the fifteenth of each month. To specify days by only one field, set the other field to *.
GX_AE_RE_KEY_NREQ. GX_AE_RE_KEY_NREQ	The AppLogic to execute when the event is triggered. Use the following format: GUIDGX-{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX?Param1=ABC&Param2=123
GX_AE_RE_KEY_MFILE. GX_AE_RE_KEY_MFILE*	The name of the file that contains the body of an email message.
GX_AE_RE_KEY_MTO. GX_AE_RE_KEY_MTO*	A comma separated list of users to send the email to.
GX_AE_RE_KEY_MHOST. GX_AE_RE_KEY_MHOST*	The name of the SMTP mail server.
GX_AE_RE_KEY_SADDR. GX_AE_RE_KEY_SADDR*	The sender’s email address.

* You must specify all of these items if sending email when the event is triggered.

Usage
Use setEvent( ) to trigger a registered event immediately. This is useful for testing purposes.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
registerEvent( )

getAppEvent( ) method in the AppLogic class

IValList interface

IAppEventMgr Interface

Application components can define events that are either triggered at a specified time or triggered explicitly. Events are stored persistently in the iPlanet Application Server, and are removed only when your application explicitly deletes them. Events are typically used to schedule routine administrative tasks, such as making back-ups or getting statistics.

iPlanet Application Server uses two new interfaces to support events:

- The IAppEventMgr interface manages application events. This interface defines methods for creating, registering, triggering, enabling, disabling, enumerating, and deleting events.
- The IAppEventObj interface represents the defined events an application supports. This interface defines methods not only for getting or setting attributes of an event, but also for adding, deleting, or enumerating actions of the event.

IAppEventMgr and IAppEventObj should be used in new or migrated applications. Existing iPlanet Application Server applications can continue using the deprecated IAppEvent interface, which supports the previous model for application events.

Attributes and Actions

For each event, you define associated attributes and actions. Attributes determine the following characteristics:

- the event's state (enabled or disabled)

- the execution mode (concurrent or serial) of the event's actions
- the time at which to trigger the event

When an event is triggered, it performs one or more of the following types of actions:

- executes a specified servlet.
- executes a specified AppLogic.
- sends an email message.

Features of Application Event Support

Support for application events includes the following:

- Added functionality
  - Execution of multiple actions of any type. (IAppEvent supports only one action of each type.)
  - In addition to executing an AppLogic and sending email, a triggered event can now execute a servlet as one of the supported action types.
  - Synchronous or asynchronous triggering of events. (IAppEvent supports only synchronous triggering.)
  - Execution of actions in the same order they are registered.
  - Execution of actions either concurrently or serially.
  - Support for passing an input IValList object to triggered events.
- Different interfaces

  Previously, application events were represented by an IValList object, and you used the IAppEvent interface to manage the events. Now an application event is represented by an IAppEventObj, and you use IAppEventMgr to manage and control the events.
- Separate actions and attributes

  Previously, attributes and actions were not distinguished. They were all treated as event properties and specified within a single IValList object. Now attributes are described by entries in one IValList object, and each action is represented by its own additional IValList object.

  IAppEventObj has methods for getting or setting attributes and for adding, deleting, or enumerating actions.

Accessing and Creating Application Events

To access an IAppEventMgr object, use the GetAppEventMgr( ) method in the GXContext class:

```
IAppEventMgr com.kivasoft.dlm.GXContext.GetAppEventMgr(
   IContext context);
```

The context parameter is an IContext object, which provides access to iPlanet Application Server services. For information about obtaining this IContext object, see the GXContext class or the IContext Interface.

After creating the IAppEventMgr object, you can create an application event (an instance of IAppEventObj) by calling createEvent( ) on the IAppEventMgr object.

Registering Events

After creating an application event, you can set its attributes and add actions using methods on the IAppEventObj. Then, register the application event by calling registerEvent( ) on the manager object.

Package

com.kivasoft

Methods

Method	Description
createEvent( )	Creates an empty application event object.
deleteEvent( )	Removes a registered event from iPlanet Application Server.
disableEvent( )	Disables a registered event.
enableEvent( )	Enables a registered event.
enumEvents( )	Enumerates through the list of registered events.
queryEvent( )	Retrieves the IAppEventObj for a registered event.
registerEvent( )	Registers a named event for use in applications.
setEvent( )	Triggers a registered event.

Related Topics

GetAppEventMgr( ) in the GXContext class,
IValList Interface (deprecated),
IAppEvent Interface (deprecated)

createEvent( )

Creates an empty application event object.

Syntax

```
public IAppEventObj createEvent(
   String pEventName)
```

pEventName. The name of the event to create.

Usage

Use createEvent( ) to create an empty IAppEventObj object. You can use methods of the IAppEventObj interface to set attributes and actions on the returned object.

Changes to the event object do not take effect until it is registered with the manager object, through a call to registerEvent( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

queryEvent( )

registerEvent( )

deleteEvent( )

Removes a registered event from iPlanet Application Server.

Syntax

```
public int deleteEvent(
   String pEventName)
```

pEventName. The name of the registered event to delete.

Usage

Use deleteEvent( ) to remove an event that is no longer required. The specified event is removed from iAS.

To temporarily stop an event from being triggered, use disableEvent( ).

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
disableEvent( )

registerEvent( )

disableEvent( )

Disables a registered event.

Syntax
```
public int disableEvent(
   String pEventName)
```

pEventName. The name of the registered event to disable.

Usage
Use disableEvent( ) to temporarily stop an event from being triggered. The event is disabled until it is enabled with enableEvent( ). To permanently remove an event from the registry, use deleteEvent( ).

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
deleteEvent( )

enableEvent( )

registerEvent( )

enableEvent( )

Enables a registered event.

Syntax
```
public int enableEvent(
   String pEventName)
```

pEventName. The name of the registered event to enable.

Usage
Use enableEvent( ) to enable an event. A given event could have been disabled in either of two ways: by a previous call to disableEvent( ) or by initially registering the event using a disabled state attribute.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
disableEvent( )

registerEvent( )

enumEvents( )
Enumerates through the list of registered events.

Syntax
```
public IEnumObject enumEvents()
```

Usage
Use enumEvents( ) to get information on all the registered events. The IEnumObject object returned by enumEvents( ) contains a set of IAppEventObj objects, one per event. Each IAppEventObj contains the attributes and actions that were assigned to the event when it was registered with registerEvent( ).

Tip
Use the methods in the IEnumObject interface to iterate through the contents of the returned IEnumObject object.

Example
The following AppLogic code shows how to use enumEvents( ) to get information on all the registered events and save it to a report:

```
// Open /tmp/report-file for writing the report
FileOutputStream outFile = null;
try {
   outFile = new FileOutputStream("/tmp/report-file");
} catch (IOException e) {
}
if (outFile == null)
   return streamResult("Can not open /tmp/report-file<br>");
```

```
ObjectOutputStream p = null;
try {
   p = new ObjectOutputStream(outFile);
} catch (IOException e) {
}
if (p == null)
    return streamResult("Cannot create ObjectOutputStream<br>");

// get appevent manager
IAppEventMgr appEventMgr = GetAppEventMgr();

// Get the Enumeration object for all registered appevents
   IEnumObject enumObj = appEventMgr.enumEvents();

// Retrieve the count of registered appevents
int count = enumObj.enumCount();
try {
    p.writeObject("Number of Registered Events: ");
    p.writeInt(count);
} catch (IOException e) {
    return streamResult("Failed to write to report file<br>");
}

enumObj.enumReset(0);

IObject eventObj = enumObj.enumNext();
while ((eventObj = enumObj.enumNext()) != null) {
    if (eventObj instanceof IAppEventObj) {
        IAppEventObj appEventObj = (IAppEventObj)eventObj;

        // print each appEvent

        // get the name of the appevent
        String name = appEventObj.getName();
        try {
            p.writeObject("\nDefinitions for AppEvent: ");
            p.writeObject(name);
            p.writeObject("\n");
        } catch (IOException e) {
               return streamResult("Failed to write to report
file<br>");
        }

        // print the attributes..
        IValList attrs = appEventObj.getAttributes();
        attrs.resetPosition();
        // Iterate through all the GXVALs in the attr s valllist
        //   and print them
             ...
             ...
             ...

        // enumerate through the actions
        enumActions = appEventObj.enumActions();
```

```
       // Retrieve the count of registered actions for this event
        count = enumActions.enumCount();
        try {
            p.writeObject("Number of Actions: ");
            p.writeInt(count);
        } catch (IOException e) {
      return streamResult("Failed to write ActionCount to report
file");
        }

        enumActions.enumReset(0);
        IObject obj ;
        while ((obj = enumActions.enumNext()) != null) {
            // print each action
            if (obj instanceof IValList) {
                IValList action = (IValList)obj;

                action.resetPosition();
                // Iterate through all the GXVALs in the vallist
                // and print them
                     ...
                     ...
                     ...
            }
        }
    }
}
```

Return Value

IEnumObject that contains the list of events, or null for failure.

Related Topics

IEnumObject interface

getEvent( )

Retrieves the IAppEventObj for a registered event.

Syntax

```
public IAppEventObj getEvent(
   String pEventName)
```

pEventName. The name of the registered event.

Usage

After calling getEvent( ), you can call methods on the returned IAppEventObj. For example, you can query the object by calling getAttributes( ) or enumActions( ), or you can modify the object by calling setAttributes( ).

Return Value

IAppEventObj for the specified event.

Related Topics

registerEvent( )

registerEvent( )

Registers a named event for use in applications.

Syntax

```
public int registerEvent(
   IAppEventObj appEventObj)
```

appEventObj. The event object whose attributes and actions have been set.

Usage

After an application event object is created with createEvent( ), you define its attributes and actions using methods of the IAppEventObj interface. Then you use registerEvent( ) to register the specified event object. Registration commits the changed attributes and actions to the server and to the registry. If an event object already exists for the given name, the existing object is deleted and replaced with the specified object.

Return Value

GXE.SUCCESS if the method succeeds.

Example

The following example shows how to define and register an application event:

```
static final java.lang.String eventName1 = "RepGenEvent";

// get the appevent manager
IAppEventMgr appEventMgr = GXContext.GetAppEventMgr(context);

// create a appevent object
IAppEventObj appEventObj = appEventMgr.createEvent(eventName1);
```

```
// create ValList for attributes.
IValList attrs = GX.CreateValList();

// set the Action Mode to Serial for serial-execution of actions.
attrs.setValInt(GXConstants.GX_AE2_RE_KEY_ACTION_MODE,
      GX_AE2_RE_SA_FLAG.GX_AE2_RE_ACTION_SERIAL);

// set the appevent time to be 05:00:00 hrs everyday
attrs.setValString(GXConstants.GX_AE2_RE_KEY_TIME, "5:0:0
*/*/*");

// set the attributes
appEventObj.setAttributes(attrs);

// create ValLists for actions (2 applogic newrequest and 2
servlet).
IValList action1 = GX.CreateValList();
IValList action2 = GX.CreateValList();
IValList action3 = GX.CreateValList();
IValList action4 = GX.CreateValList();

// set action1 to run an appLogic
action1.setValString(GXConstants.GX_AE2_RE_KEY_NREQ,
       "GUIDGX-{620CB09B-1A1D-1315-AD23-0800207B918B}");

// set action1 to run another appLogic
action2.setValString(GXConstants.GX_AE2_RE_KEY_NREQ,
       "GUIDGX-{60A36CC0-9F69-11D2-9907-0060973797BF}");

// set action2 to run a servlet
action3.setValString(GXConstants.GX_AE2_RE_KEY_SERVLET,
       "TxReportServlet");

// set action3 to run another servlet
action4.setValString(GXConstants.GX_AE2_RE_KEY_SERVLET,
       "SummaryReportServlet");

// add actions in the order we want them to be executed
appEventObj.addAction(action1);
appEventObj.addAction(action2);
appEventObj.addAction(action3);
appEventObj.addAction(action4);

// Register the event
if (appEventMgr.registerEvent(eventName1, appEventObj) !=
GXE.SUCCESS)
   return streamResult("Cannot register RepGenEvent<br>");
```

Related Topics

enableEvent( )

setEvent( )

triggerEvent( )

Triggers a registered event.

Syntax

```
public int triggerEvent(
   String pEventName,
   IValList pValList,
   boolean syncFlag)
```

pEventName. The name of the event to trigger.

pValList. The IValList object that specifies the input that is passed to the triggered event and its actions.

syncFlag. The boolean flag that indicates whether the event is to be triggered synchronously (value is true) or asynchronously (value is false).

Usage

Use triggerEvent( ) to trigger a registered event. When you specify the pValList parameter, a copy of this IValList object is passed as input to all actions registered with the application event.

If the action is ...	Then pValList is ...
an AppLogic.	passed as input to that AppLogic.
an email message.	simply ignored.
a servlet.	passed to the servlet as the valIn of the underlying AppLogic.

Use the syncFlag parameter to determine synchronous or asynchrous execution. Typical usage is to set syncFlag to false, which provides asynchronous execution and better application performance. When syncFlag is false, the event is triggered, and the method call returns immediately, without waiting for the actions to finish executing.

If syncFlag is true, then the method call does not return immediately. Instead, the call blocks until the event is triggered and all actions have executed. In some cases, it may be desirable for actions to finish executing before returning control to the application.

Actions are triggered in the same order in which they were added to the application event object.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
registerEvent( )

IAppEventObj Interface

See the IAppEventMgr interface for details on IAppEventObj.

Package

com.kivasoft

Methods

Method	Description
addAction( )	Appends an action to an ordered list of actions.
deleteActions( )	Deletes all actions added to this IAppEventObj.
enumActions( )	Enumerates the actions added to this IAppEventObj.
getAttributes( )	Retrieves the list of attributes of an IAppEventObj.
getName( )	Retrieves the name of the IAppEventObj.
setAttributes( )	Sets a list of attribute values for the IAppEventObj.

Related Topics

IAppEventMgr interface

addAction( )

Appends an action to an ordered list of actions.

Syntax

```
public int addAction(
    IValList action)
```

action. The input IValList object that defines the action to add. When an event is triggered, actions are executed in the same order in which they were added. The entries in this IValList object vary from one action type to another.

The keys and values you can specify are as follows. Note that string constants are available in the class com.kivasoft.types.GXConstants.

For AppLogics:

Key	Value
GXConstants.GX_AE2_RE_KEY_NREQ	The AppLogic to execute when the event is triggered. Use the following format: GUIDGX-{*XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX*?*Param1=ABC*&*Param2=123*. The parameters and their values are passed as input to the events.

For email:

To send email when the event is triggered, all of the following items must be specified.

Key	Value
GXConstants. GX_AE2_RE_KEY_MFILE	The name of the file that contains the body of an email message.
GXConstants. GX_AE2_RE_KEY_MTO	A comma separated list of users to send the email to.
GXConstants. GX_AE2_RE_KEY_MHOST	The name of the SMTP mail server.
GXConstants. GX_AE2_RE_KEY_SADDR	The sender's email address.

For servlets:

Key	Value
GXConstants. GX_AE2_RE_KEY_SERVLET	The name of the servlet to be executed when the event is triggered. Use the following format: *appName*/*ServletName*?*Param1=ABC*&*Param2=123*. Parameters and their values are passed as input to the events. The only required item is the servlet name. The application name and parameters are optional.

Usage

Use the addAction( ) method after creating an application event object (by calling createEvent( ) on the IAppEventMgr object). After you change an event (for example, by adding or deleting actions or by setting attributes), you must register the event in order for the changes to take effect.

To list the added actions, use enumActions( ). To delete all actions, use deleteActions( ).

Return Value

GXE.SUCCESS if the method succeeds.

deleteActions( )

Deletes all actions added to this IAppEventObj.

Syntax

```
public int deleteActions()
```

Usage

Use this method to remove all actions associated with this IAppEventObj.

Return Value

GXE.SUCCESS if the method succeeds.

enumActions( )

Enumerates the actions added to this IAppEventObj.

Syntax
```
public IEnumObject enumActions()
```

Usage
Use this method to obtain a list of actions that have been added to this IAppEventObj. Each entry in the returned IEnumObject is an IValList object representing an action.

Return Value
IEnumObject that contains the list of actions, or null for failure.

getAttributes( )
Retrieves the list of attributes of an IAppEventObj.

Syntax
```
public IValList getAttributes()
```

Usage
Call this method after calling setAttributes( ).

Return Value
IValList object that contains the list of attributes, or null for failure.

Related Topics
setAttributes( )

getName( )
Retrieves the name of the IAppEventObj.

Syntax
```
public String getName()
```

pName. A pointer to an input buffer.

nName. The size of the input buffer.

Usage
The name of an IAppEventObj is set by calling createEvent( ) on the IAppEventMgr object. After creating an application event object, use the getName( ) method to retrieve the name.

Return Value

A string representing the name of the application event object, or null for failure.

setAttributes( )

Sets a list of attribute values for the IAppEventObj.

Syntax

```
public int setAttributes(
   IValList attrList)
```

attrList. The input IValList object that specifies the attributes. The keys and values you can specify are as follows. Note that string constants are available in the class com.kivasoft.types.GXConstants.

- GXConstants.GX_AE2_RE_KEY_STATE

  A variable that specifies the initial state of the event. This key is optional and has the following possible values:

  - GXConstants.GX_AE2_RE_EVENT_DISABLED
  - GXConstants.GX_AE2_RE_EVENT_ENABLED (the default)

- GXConstants.GX_AE2_RE_KEY_TIME

  An optional key that specifies the time at which the event will be triggered. Use the following format:

  - hh:mm:ss W / DD / MM
  - hh: 0 -23
  - mm: 0 - 59
  - ss: 0 - 59
  - W (day of the week): 0 - 6 with 0 = Sunday.
  - DD (day of the month): 1 - 31

- MM (month): 1 - 12

Each of these fields may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign indicating an inclusive range. For example, 2, 5 - 7:0:0 5/*/* means the event is triggered at 2 AM, 5AM, 6 AM and 7 AM every Friday.

The specification of days can be made by two fields: day of the month (DD) and day of the week (W). If both are specified, both take effect. For example, 1:0:0 1/15/* means the event is triggered at 1 AM every Monday, as well as on the fifteenth of each month. To specify days by only one field, set the other field to *.

- GXConstants.GX_AE2_RE_KEY_ACTION_MODE

  An optional key that specifies whether actions are to be executed concurrently (at the same time) or in series (one after another). In serial execution, each action finishes executing before the next one starts, and execution occurs in the same order in which the actions were added.

  This key has the following possible values:

  - GXConstants.GX_AE2_RE_ACTION_SERIAL
  - GXConstants.GX_AE2_RE_ACTION_CONCURRENT (the default)

Usage

Use the setAttributes( ) method after creating an application event object (by calling createEvent( ) on the IAppEventMgr object). After you change an event (for example, by adding or deleting actions or by changing attributes), you must register the event in order for the changes to take effect.

Tip

None of the attributes are required to be set. The default state is enabled, and the default action mode is concurrent.

To retrieve the list of attributes that are set, use getAttributes( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

getAttributes( )

IBuffer Interface *(deprecated)*

The IBuffer interface is deprecated and is provided for backward compatibility only. New applications developed according to the servlet-JSP programming model do not need the functionality provided by IBuffer.

The IBuffer interface represents a block of memory. Input arguments and output value(s) of methods are sometimes stored in IBuffer objects. For example, the get**( ) methods in the IQuery interface return values in an IBuffer object.

IBuffer provides methods for specifying and obtaining the size of the memory block, obtaining its starting address, and copying data to it.

To create an instance of the IBuffer interface, use the GX.CreateBuffer( ) method. To create and initialize an IBuffer object with string contents, use GX.CreateBufferFromString( ).

Package

com.kivasoft

Methods

Method	Description
alloc( )	Specifies the size of the memory block, in bytes.
getAddress( )	Returns a copy of the buffer contents.
getSize( )	Returns the size of the memory block, in bytes.
setData( )	Copies data to a memory block.

alloc( )

Specifies the size of the memory block, in bytes.

Syntax

```
public int alloc(
   int nSize)
```

nSize. Size of the memory block, in bytes.

Usage

After creating a memory buffer with the GX.CreateBuffer( ) function, use alloc( ) to specify its size.

Subsequent calls to getSize( ) return the size that AppLogic specified when it called alloc( ).

Rules

- If the AppLogic creates its own new IBuffer object, it must first specify the size of the memory block by calling alloc( ) before using other methods in the interface.
- AppLogic can call alloc( ) only once.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
String str = "Hello World";
byte tmp[] = new byte[12];
str.getBytes(0, 11, tmp, 0);

IBuffer buff;

buff = GX.CreateBuffer();
buff.alloc(128);
buff.setData(tmp, 12);

// The following example is an alternate to the
// previous one. It uses GX.CreateBufferFromString()
// instead of GX.CreateBuffer().

IBuffer buff;
buff = GX.CreateBufferFromString("Hello World");
```

Related Topics

getAddress( )

getAddress( )

Returns a copy of the buffer contents.

Syntax

```
public byte[] getAddress()
```

Usage
Use getAddress( ) to get a copy of the contents in a buffer.

Return Value
A copy of the buffer contents.

Related Topics
alloc( )

getSize( )

Returns the size of the memory block, in bytes.

Syntax
```
public int getSize()
```

Usage
Use getSize( ) to determine the length of the memory buffer that the AppLogic specified when it called alloc( ).

Rule
Before calling getSize( ), AppLogic must first specify the size of the memory block by calling alloc( ).

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
getAddress( )

setData( )

setData( )

Copies data to a memory block.

Syntax
```
public int setData(
   byte[] pData,
   int nDataLen)
```

pData. The data to copy to the memory buffer.

nDataLen. The length, in bytes, of the data to copy to the memory buffer.

Usage

Use setData( ) to copy data to a memory buffer. The buffer can then be passed to a method, such as the setValPieceByOrd( ) method in the ITable interface, that accepts data values in a buffer object.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
String str = "Hello World";
byte tmp[] = new byte[12];
str.getBytes(0, 11, tmp, 0);

IBuffer buff;

buff = GX.CreateBuffer();
buff.alloc(128);
buff.setData(tmp, 12);

table.setValPieceByOrd(1, buff, 12);
```

Related Topics

getAddress( )

getSize( )

ICallableStmt Interface *(deprecated)*

ICallableStmt is deprecated and is provided for backward compatibility only. New applications should use the java.sql.ICallableStatement interface from the JDBC Core API.

The ICallableStmt interface provides a standard way to call stored procedures in any database server. A stored procedure is a block of SQL statements stored in a database. Stored procedures provide centralized code for manipulating data and reduce the amount of data that needs to be sent to the client side of an application. They are typically used to execute database operations, for example, modify, insert, or delete records.

To call a stored procedure from an AppLogic, use the ICallableStmt object. The ICallableStmt interface defines methods for executing a stored procedure or function, and setting and getting parameter values to and from a stored procedure.

To create an instance of the ICallableStmt interface, use prepareCall( ) in the IDataConn interface.

Package

com.kivasoft

Methods

Method	Description
close( )	Releases the callable statement.
execute( )	Executes the stored procedure called by the ICallableStatement object.
executeMultipleRS( )	Executes a stored procedure, called by the ICallableStmt object, that can return multiple result sets.
getMoreResults( )	Checks if there is a result set to retrieve. This method is valid only if you used executeMultipleRS( ), not execute( ), to execute a stored procedure called by the ICallableStmt object.
getParams( )	Returns the value of the stored procedure's output parameter or parameters.
getResultSet( )	Retrieves a result set. This method is valid only if you used executeMultipleRS( ) (instead of execute( )) to execute a stored procedure called by the ICallableStmt object.
setParams( )	Specifies the parameter values to pass to the stored procedure.

Related Topics

prepareCall( ) in the IDataConn Interface (deprecated)

close( )

Releases the callable statement.

Syntax

```
public int close()
```

Usage

Use close( ) to release a callable statement object after the AppLogic has finished processing the results returned by the stored procedure. You must release the callable statement object so that the data connection is available to process other commands.

Return Value

GXE.SUCCESS if the method succeeds.

execute( )

Executes the stored procedure called by the ICallableStmt object.

Syntax

```
public IResultSet execute(
   int flags,
   IValList params,
   ITrans trans,
   IValList props);
```

flags .

- For synchronous operations, the default, specify 0 (zero) or GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_SYNC.
- For asynchronous operations, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC.

params . IValList object that contains parameters to pass to the callable statement. If you use setParams( ) instead to specify the parameters, specify null here.

trans . ITrans object that contains the transaction associated with this callable statement, or null for no transaction.

props . IValList object that contains properties, or null for no properties. This parameter applies only if the callable statement returns a result set. Informix stored procedures, for example, return out parameter values only as a result set. Sybase, DB2, and MS SQL Server stored procedures also support the return of a result set. Multiple result sets, however, is not supported.

After instantiating an object of the IValList interface, set any of the following properties:

- RS_BUFFERING turns on result set buffering when set to “TRUE” or “YES.”

- RS_INIT_ROWS specifies the initial size of the buffer, in number of rows. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED and result set buffering will be turned off.
- RS_MAX_ROWS specifies the maximum number of rows for the buffer. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED and result set buffering will be turned off.
- RS_MAX_SIZE specifies the maximum number of bytes for the buffer.

If RS_BUFFERING is enabled and if the optional parameters are not specified, the global values in the registry are used instead.

Usage

Use execute( ) to run a callable statement that has been created with prepareCall( ) in the IDataConn interface. If the stored procedure called by the ICallableStmt object can return multiple result sets, use executeMultipleRS( ) instead.

If the stored procedure called by the ICallableStmt object contains parameters, instantiate an IValList object and use setValString( ) or setValInt( ) in the IValList interface to specify the parameter values to pass to the stored procedure.

After creating and setting up the IValList object, pass it to execute( ) or setParams. If you use setParams( ) to pass parameters to the stored procedure, specify NULL for the params parameter in execute( ).

Rule

When accessing a stored procedure on Sybase or MS SQL Server, input parameter names specified in the call must be prefixed with the ampersand (&) character, for example, &param1. Other database drivers accept the ampersand, as well, as the colon (:) character. For all database drivers, input/output and output parameter names are prefixed with the colon (:) character, for example, :param2.

Return Value

IResultSet object, or null for failure (such as an invalid parameter).

Example

```
//Write command to call stored procedure
String theProc = "{call myProc1(&p1, :p2)}";
IQuery myquery;
myquery = createQuery();
myquery.setSQL(theProc);
```

```
//Prepare the callable statement for execution
ICallableStmt myStmt;
myStmt = conn_rtest.prepareCall(0, myquery, null, null);

//Set the in parameter value
IValList inParams = GX.CreateValList();
inParams.setValInt("&p1",6210603);
//Set the out parameter value to any integer
inParams.setValInt(":p2", 0);

//Run the callable statement
myStmt.execute(0, inParams, null, null);

//Get the stored procedure's output value
outParams = myStmt.getParams(0);
```

Related Topics

prepareCall( ) in the IDataConn Interface (deprecated)

getParams( )

setParams( )

executeMultipleRS( )

Executes a stored procedure, called by the ICallableStmt object, that can return multiple result sets.

Syntax

```
public int executeMultipleRS(
   int dwFlags,
   IValList pParams,
   ITrans pTrans,
   IValList pProps);
```

dwFlags.

Specify 0.

pParams. IValList object that contains parameters to pass to the callable statement. If no parameters are required, pass in an empty IValList. If you use setParams( ) instead to specify the parameters, specify null here.

pTrans. ITrans object that contains the transaction associated with this callable statement, or null for no transaction.

pProps. IValList object that contains properties, or null for no properties.

After instantiating an object of the IValList interface, set any of the following properties:

- RS_BUFFERING turns on result set buffering when set to “TRUE” or “YES.”
- RS_INIT_ROWS specifies the initial size of the buffer, in number of rows. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED and result set buffering will be turned off.
- RS_MAX_ROWS specifies the maximum number of rows for the buffer. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED and result set buffering will be turned off.
- RS_MAX_SIZE specifies the maximum number of bytes for the buffer.

If RS_BUFFERING is enabled and if the optional parameters are not specified, the global values in the registry are used instead.

Usage

Use executeMultipleRS( ) to run a callable statement that returns multiple result sets. The callable statement should already have been created with prepareCall( ) in the IDataConn interface.

If the stored procedure called by the ICallableStmt object contains parameters, instantiate an IValList object and use setValString( ) or setValInt( ) in the IValList interface to specify the parameter values to pass to the stored procedure.

After creating and setting up the IValList object, pass it to executeMultipleRS( ) or setParams. If you use setParams( ) to pass parameters to the stored procedure, specify NULL for the pParams parameter in executeMultipleRS( ).

Rule

When accessing a stored procedure on Sybase or MS SQL Server, input parameter names specified in the call must be prefixed with the ampersand (&) character, for example, &param1. Other database drivers accept the ampersand, as well, as the colon (:) character. For all database drivers, input/output and output parameter names are prefixed with the colon (:) character, for example, :param2.

Tip

The difference between execute( ) and executeMultipleRS( ) is that execute( ) can return only a single result set. If you’re not sure how many results sets, if any, a stored procedure returns, use executeMultipleRS( ).

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
//Write command to call stored procedure
String proccall = "{call multiRSproc()}";
IQuery qryobj = GX.createQuery();
qryobj.setSQL(proccall);

//Prepare the callable statement for execution
ICallableStmt stmt = connobj.PrepareCall(0, qryobj, null, null);

//Create the IValList to pass to the stored procedure
IValList inparms = GX.CreateValList();

//Run the callable statement
stmt.executeMultipleRS(0, inparms, null, null);

//Check if there is a result set
if (stmt.getMoreResults())
{
   //Get the result set
   IResultSet rsobj = stmt.getResultSet();
   while (rsobj.fetchNext()) {
      if (rsobj.getValueInt(1) == 100) {
         System.out.println("Found record 100");
      }
   }
}
//Check if there is another result set
if (stmt.getMoreResults()) {
   rsobj = stmt.getResultSet();
   while (rsobj.fetchNext()) {
      if (rsobj.getValueString(2) == 'George') {
         System.out.println("Found record George");
      }
   }
}
```

Related Topics

prepareCall( ) in the IDataConn Interface (deprecated)

getMoreResults( )

getResultSet( )

getMoreResults( )

Checks if there is a result set to retrieve. This method is valid only if you used executeMultipleRS( ) (instead of execute( )) to execute a stored procedure called by the ICallableStmt object.

Syntax

```
public boolean getMoreResults()
```

Usage

If you used executeMultipleRS( ) to execute a stored procedure that returns multiple results sets, use getMoreResults( ) in conjunction with getResultSet( ) to check if there is a result set before retrieving it.

If there is a current result set with unretrieved rows, getMoreResults( ) discards the current result set and makes the next result set available.

Return Value

True if there is a result set.

Example

```
//Run the callable statement
stmt.executeMultipleRS(0, inparms, null, null);

//Check if there is a result set
if (stmt.getMoreResults())
{
   //Get the result set
   IResultSet rsobj = stmt.getResultSet();
   while (rsobj.fetchNext()) {
      if (rsobj.getValueInt(1) == 100) {
         System.out.println("Found record 100");
      }
   }
}
//Check if there is another result set
if (stmt.getMoreResults()) {
   rsobj = stmt.getResultSet();
   while (rsobj.fetchNext()) {
      if (rsobj.getValueString(2) == 'George') {
          System.out.println("Found record George");
      }
   }
}
```

Related Topics

prepareCall( ) in the IDataConn Interface (deprecated)

executeMultipleRS( )

getResultSet( )

getParams( )

Returns the value of the stored procedure's output parameter or parameters.

Syntax

```
public IValList getParams(
   int dwFlags);
```

dwFlags. Specify 0 (zero).

Usage

Some stored procedures return output parameters. If the stored procedure your callable statement executes returns output parameters, use getParams( ) to get the values.

The getParams( ) method returns the values in an IValList object. The key names associated with the values are the parameter names as specified in the query that was passed to the prepareCall( ) method

Tip

Informix stored procedures return output parameters in a result set. This result set is returned by execute( ) or executeMultipleRS( ). The getParams( ) method, therefore, does not apply to Informix stored procedures.

Return Value

IValList object, or null for failure.

Related Topics

prepareCall( ) in the IDataConn Interface (deprecated)

execute( )

setParams( )

getResultSet( )

Retrieves a result set. This method is valid only if you used executeMultipleRS( ) (instead of execute( )) to execute a stored procedure called by the ICallableStmt object.

Syntax

```
public IResultSet getResultSet()
```

Usage

If you used executeMultipleRS( ) to execute a stored procedure that returns multiple results sets, use getResultSet( ) in conjunction with getMoreResults( ) to retrieve the results sets.

Return Value

IResultSet object, or null if there is no result set.

Example

```
//Run the callable statement
stmt.executeMultipleRS(0, inparms, null, null);

//Check if there is a result set
if (stmt.getMoreResults())
{
   //Get the result set
   IResultSet rsobj = stmt.getResultSet();
   while (rsobj.fetchNext()) {
      if (rsobj.getValueInt(1) == 100) {
         System.out.println("Found record 100");
      }
   }
}
//Check if there is another result set
if (stmt.getMoreResults()) {
   rsobj = stmt.getResultSet();
   while (rsobj.fetchNext()) {
      if (rsobj.getValueString(2) == 'George') {
          System.out.println("Found record George");
      }
   }
}
```

Related Topics

prepareCall( ) in the IDataConn Interface (deprecated)

executeMultipleRS( )

getMoreResults( )

setParams( )

Specifies the parameter values to pass to the stored procedure.

Syntax

```
public int setParams(
   int dwFlags,
   IValList pParams);
```

dwFlags. Specify 0 (zero). For internal use only.

pParams. IValList object that contains the parameters to pass to the stored procedure. You must set all parameters required by the stored procedure. If you don't, a runtime error will occur when execute( ) is called. If you use setParams( ), specify NULL for the pParams parameter in execute( ).

Usage

If the stored procedure the callable statement executes accepts input parameters, use setParams( ) to pass the parameter or parameter values. The alternative is to pass the parameter values with the execute( ) method. Parameters passed to execute( ) supersede parameters specified with setParams( ).

For both setParams( ) and execute( ), you pass the parameter values in an IValList object.

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

prepareCall( ) in the IDataConn Interface (deprecated)

execute( )

getParams( )

ICallerContext Interface

The ICallerContext interface manages programmatic security from within a servlet. ICallerContext provides two methods that override equivalent methods in the standard javax.ejb.EJBContext interface.

From within EJBs, do not use ICallerContext. Use javax.ejb.EJBContext instead, to maintain the portability of the bean.

Typically ICallerContext is used together with the IServerContext interface.

Package

com.netscape.server

Methods

The ICallerContext interface is described as follows:

```
public interface ICallerContext
{
    public java.security.Identity getCallerIdentity();
    public boolean isCallerInRole(java.security.Identity role);
}
```

getCallerIdentity( ) returns the Identity object that identifies the caller.

The isCallerInRole( ) method returns true if the caller has the given role, where the role parameter describes the java.security.Identity of the role to be tested.

Neither of the specifications for servlets and EJBs describe a standard way of creating an instance of java.security.Identity. To create an instance within a iPlanet Application Server 6.0 application, use the following method from the IServerContext interface:

```
public java.security.Identity createIdentityByString(
    String identity)
```

Examples

Example 1

The following sample code demonstrates the use of programmatic security from a servlet:

```
// from within a servlet
com.netscape.server.IServerContext iplanetCtx;
ServletContext ctx = getServletContext();
iplanetCtx = (com.netscape.server.IServerContext) ctx;

com.netscape.server.ICallerContext callerCtx;
```

```
callerCtx = iplanetCtx.getCallerContext();

Identity id = callerCtx.getCallerIdentity();

if(id == null)
    System.out.println("Servlet invoked by anonymous user");
else
    System.out.println("Servlet invoked by user " + id.getName());
```

Example 2

The following sample code demonstrates the use of programmatic security from a servlet:

```
// from within a servlet
com.netscape.server.IServerContext iplanetCtx;
ServletContext ctx = getServletContext();
iplanetCtx = (com.netscape.server.IServerContxt) ctx;

com.netscape.server.ICallerContext callerCtx;
Identity managers =
iplanetCtx.createIdentityByString("managers");

if(callerCtx.isCallerInRole(managers))
  // caller is a member of the group managers
else
  // caller is not a manager
```

Example 3

The following sample code demonstrates the use of programmatic security from a bean.

Both SessionContext and EntityContext inherit from javax.ejb.EJBContext. The container will invoke setSessionContext( ) or setEntityContext( ) on your bean and provide you with an instance before any business methods are executed.

```
// within a bean

javax.ejb.SessionContext m_ctx;
```

```
// method on javax.ejb.EJBContext
Identity id = m_ctx.getCallerIdentity();
```

In addition, the isCallerInRole() method, available from ICallerContext and EJBContext, can be used to test for membership.

Related Topics

com.kivasoft.applogic.AppLogic class (deprecated),
com.kivasoft.dlm.GXContext class,
com.kivasoft.IContext Interface,
com.netscape.server.IServerContext Interface

javax.ejb.EJBContext interface

IColumn Interface *(deprecated)*

IColumn is deprecated and is provided for backward compatibility only. New applications should use the interfaces java.sql.DatabaseMetaData or java.sql.ResultSetMetaData from the JDBC Core API.

The IColumn interface represents a column definition in a table. IColumn provides methods for obtaining descriptive information about a table column from the database catalog, which contains the column definition. Column attributes include the column name, precision, scale, size, table, and data type.

IColumn is part of the Data Access Engine (DAE) service.

To create an instance of this interface, use one of the following methods:

- getColumn( ) or getColumnByOrd( ) in the IHierResultSet Interface (deprecated)
- getColumn( ), getColumnByOrd( ), or enumColumns( ) in the ITable Interface (deprecated)
- getColumn( ), getColumnByOrd( ), or enumColumns( ) in the IResultSet Interface (deprecated)

The following example shows creating an IColumn object:

```
ITable tbl = conn.getTable("Products");
```

```
IColumn col = tbl.getColumnByOrd(1);
```

Package

com.kivasoft

Methods

Method	Description
getName( )	Returns the name of the column or alias.
getNullsAllowed( )	Returns true if null values are allowed in the column.
getPrecision( )	Returns the precision, which is the maximum length or maximum number of digits, of the column.
getScale( )	Returns the scale, which is the number of digits to the right of the decimal point, of the column of type double.
getSize( )	Returns the maximum length, in number of bytes, allowed for a value in this column.
getTable( )	Returns the table object in which this column exists.
getType( )	Returns the data type of the column.

Example 1

The following example shows how to iterate through a table to get the names and types of the columns:

```
String htmlString;
IColumn col;
ITable tbl = conn.getTable("Products");

htmlString += "<h2>Products Table</h2>";
tbl.enumColumnReset();

while ((col = tbl.enumColumns()) != null) {
   htmlString += "Column Name = ";
   htmlString += col.getName();
   htmlString += ", Column Type = ";
   htmlString += col.getType();
   htmlString += "<br>";
};
return result(htmlString)
```

Example 2

The following example goes through all columns in a Products table and constructs an HTML string showing catalog information about each column:

```
ITable tbl = conn.getTable("Products");
htmlString += "<h2>Products Table:</h2>";

tbl.enumColumnReset();
IColumn col = tbl.enumColumns();
int loopcnt = 1;

while (col != null) {
   htmlString += "Table Name = ";
   htmlString += col.getTable().getName();
   htmlString += "<br>";
   htmlString += "Column Name=";
   htmlString += col.getName();
   htmlString += "<br>";
   htmlString += ", Column Type=";
   htmlString += col.getType();
   htmlString += "<br>";
   htmlString += "Nulls Allowed (T/F) = ";
   htmlString += col.getNullsAllowed();
   htmlString += "<br>";
   htmlString += "Max Size = ";
   htmlString += col.getSize();
   htmlString += "<br>";
   htmlString += "Precision = ";
   htmlString += col.getPrecision();
   htmlString += "<br>";

   if (col.getType() == GX_DA_DATA_TYPES.GX_DA_TYPE_DOUBLE){
      htmlString += "Scale = ";
      htmlString += col.getScale();
      htmlString += "<br>";
   }
   htmlString += "<br>";

   col = tbl.enumColumns();
   loopcnt++;
   };
   return result(htmlString + stdResFooter());
   }
}
```

Related Topics

getColumn( ) or getColumnByOrd( ) in the IHierResultSet Interface (deprecated)

getColumn( ), getColumnByOrd( ), or enumColumns( ) in the ITable Interface (deprecated)

getColumn( ), getColumnByOrd( ), or enumColumns( ) in the IResultSet Interface (deprecated)

getName( )

Returns the name of the column or alias.

Syntax

```
public String getName()
```

Usage

Use getName( ) when the name of the column is unknown and is required for subsequent operations.

Tips

- For computed columns in a query, specify aliases so that using getName( ) returns the alias name. Otherwise, the column can be identified only by ordinal position.
- Do not rely on the case of the returned name. It might be all uppercase or mixed case, depending on the database.

Return Value

String representing the name of the column, or null for failure (such as insufficient memory).

Example

The following example shows how to iterate through a table to get the names of columns:

```
ITable tbl = conn.getTable("Products");
tbl.enumColumnReset();

htmlString += "<h2>Products Table:</h2>";
IColumn = tbl.enumColumns();
int loopcnt;
loopcnt = 1;

while (col != null) {
   htmlString += "Column Name=";
   htmlString += col.getName();
   htmlString += "<br>";
```

```
   col = tbl.enumColumns();
     loopcnt++;
     };
   return result(htmlString + stdResFooter());
   }
}
```

getNullsAllowed( )

Determines whether null values are allowed in the column.

Syntax

```
public boolean getNullsAllowed()
```

Usage

A column may require data values. Use getNullsAllowed( ) if this information is unknown to determine, for subsequent operations, whether nulls are allowed or not.

Tip

For numeric columns that allow NULLs, the value is usually zero (0) in the column if a NULL is inserted. For more information, see your database vendor's documentation.

Return Value

True if the column allows nulls, or false if the column is defined as NOT NULL (requiring data values).

Example

The following example shows how to iterate through a table and return the column names, as well as whether null values are allowed in each column:

```
ITable tbl = conn.getTable("Products");
tbl.enumColumnReset();

htmlString += "<h2>Products Table:</h2>";

IColumn col = tbl.enumColumns();
int loopcnt;
loopcnt = 1;

while (col != null) {
   htmlString += "Column Name=";
```

```
   htmlString += col.getName();
   htmlString += "<br>";
   htmlString += "Nulls Allowed (T/F) = ";
   htmlString += col.getNullsAllowed();
   htmlString += "<br>";
   htmlString += "<br>";
   col = tbl.enumColumns();
      loopcnt++;
      };
   return result(htmlString + stdResFooter());
   }
}
```

getPrecision( )

Returns the precision, which is the maximum length or maximum number of digits, of the column.

Syntax

```
public int getPrecision()
```

Usage

Use getPrecision( ) when the precision of the column is unknown and is required for subsequent operations.

Return Value

An int value representing the maximum length or maximum number of digits of the column, or zero for failure (such as the precision is unknown).

Example

The following example shows how to iterate through a table and return the column names, as well as the precision value of each column:

```
ITable tbl = conn.getTable("Products");
tbl.enumColumnReset();

htmlString += "<h2>Products Table:</h2>";

IColumn col = tbl.enumColumns();
int loopcnt;
loopcnt = 1;

while (col != null) {
   htmlString += "Column Name=";
```

```
    htmlString += col.getName();
    htmlString += "<br>";
    htmlString += "Precision = ";
    htmlString += col.getPrecision();
    htmlString += "<br>";
    htmlString += "<br>";
    col = tbl.enumColumns();
    loopcnt++;
    };
    return result(htmlString + stdResFooter());
    }
}
```

getScale( )

Returns the scale, which is the number of digits to the right of the decimal point, of a column of type double.

Syntax

```
public int getScale()
```

Usage

Use getScale( ) when the scale of the column is unknown and is required for subsequent operations.

Rules

- Use getScale( ) with numeric columns, including SQL DECIMAL, NUMERIC, and FLOAT data types.
- The value returned from getScale( ) depends on the data type of the column. For example, it returns zero (0) for integers. For more information, see your database server documentation.
- For computed columns in a result set, the value returned from getScale( ) depends on the data type of the evaluated expression.

Return Value

An int value representing the fixed number of digits to the right of the decimal point, or zero if unknown or not applicable.

getSize( )

Returns the maximum length, in number of bytes, allowed for a value in this column.

Syntax

```
public int getSize()
```

Usage

Use getSize( ) when the maximum allowable length of the column is unknown and is required for subsequent operations. Note that getSize( ) does not return the actual size of data in the column.

Rules

- The value returned from getSize( ) depends on the data type of the column. For more information, see your database server documentation.
- For computed columns in a result set, the value returned from getSize( ) depends on the data type of the evaluated expression.

Return Value

An int value representing the maximum length of the column, or zero for failure.

Example

The following example shows how to iterate through a table and return the column names, as well as the maximum allowable length of each column:

```
ITable tbl = conn.getTable("Products");
tbl.enumColumnReset();

htmlString += "<h2>Products Table:</h2>";

IColumn col = tbl.enumColumns();
int loopcnt;
loopcnt = 1;

while (col != null) {
   htmlString += "Column Name=";
   htmlString += col.getName();
   htmlString += "<br>";
   htmlString += "Max Size = ";
   htmlString += col.getSize();
   htmlString += "<br>";
   htmlString += "<br>";
```

```
   col = tbl.enumColumns();
   loopcnt++;
   };
   return result(htmlString + stdResFooter());
   }
}
```

getTable( )

Returns the table object in which this column exists.

Syntax

```
public ITable getTable()
```

Usage

Use getTable( ) when the table definition of the column is unknown and is required for subsequent operations. For result set columns, this method returns a table object, which is a description of the columns in the result set.

Return Value

ITable object representing the table, or null for failure.

Example

```
// Walk through all columns in Products table and construct
// an HTML string showing the table name for each column
ITable tbl = conn.getTable("Products");
tbl.enumColumnReset();
htmlString += "<h2>Products Table:</h2>";
IColumn col = tbl.enumColumns();
int loopcnt;
loopcnt = 1;
while (col != null) {
   htmlString += "Table Name = ";
   htmlString += col.getTable().getName();
   htmlString += "<br>";
   htmlString += "Column Name=";
   htmlString += col.getName();
   htmlString += "<br>";
   htmlString += "<br>";
   col = tbl.enumColumns();
   loopcnt++;
   };
```

```
   return result(htmlString + stdResFooter());
   }
}
```

Related Topics

ITable Interface (deprecated)

getType( )

Returns the data type of the column.

Syntax

```
public int getType()
```

Use getType( ) when the data type of the column is unknown and is required for subsequent operations.

Return Value

An int value corresponding to one of the static variables, listed next, in the GX_DA_DATA_TYPES class.

GX_DA_TYPE_ERROR	GX_DA_TYPE_TIME
GX_DA_TYPE_BINARY	GX_DA_TYPE_DOUBLE
GX_DA_TYPE_DATETIME	GX_DA_TYPE_LONG
GX_DA_TYPE_DATE	GX_DA_TYPE_STRING

Example

```
// Pulls a particular column value from the result set
private static String getColumnString(IResultSet res, String
colName) {

// Input parameter error checking
if((rs==null)||(rs.getRowNumber()==0)||(colName==null)) return
null;
```

```
// Pulls a particular column value from the result set
int colOrd=rs.getColumnOrdinal(colName);
IColumn col;

if((col=rs.getColumnByOrd(colOrd))==null) return null;
String valStr=null;

// Switch type
switch(col.getType()) {
   case GX_DA_DATA_TYPES.GX_DA_TYPE_STRING:
      valStr = rs.getValueString(colOrd);
      break;
   case GX_DA_DATA_TYPES.GX_DA_TYPE_LONG:
      valStr=String.valueOf(rs.getValueInt(colOrd));
      break;
   case GX_DA_DATA_TYPES.GX_DA_TYPE_DATE:
   case GX_DA_DATA_TYPES.GX_DA_TYPE_DATETIME:
   case GX_DA_DATA_TYPES.GX_DA_TYPE_TIME:
      valStr = rs.getValueDateString(colOrd);
      break;
   case GX_DA_DATA_TYPES.GX_DA_TYPE_DOUBLE:
      valStr = String.valueOf(rs.getValueDouble(colOrd));
      break;
   default: // Unknown type, so error out
   // Fall thru to the return. Note that valStr is null;
};
   return valStr;
```

IContext Interface

In iPlanet Application Server applications, some public methods require an IContext object as a parameter. The IContext interface itself has no public methods. You do not create or destroy IContext objects, but you can obtain them in several ways.

Package

com.kivasoft

Examples

The following examples describe different ways to obtain a context.

Example 1

From an AppLogic, an instance of this context is available as a member variable `context` within the superclass com.kivasoft.applogic.Applogic. The following code is used from within an AppLogic method:

```
com.kivasoft.IContext kivaContext;
```

```
kivaContext = this.context;
```

Example 2

From a servlet, the standard servlet context can be cast to IServerContext, and from there, a com.kivasoft.IContext instance can be obtained. The servlet code would look like this:

```
ServletContext ctx = getServletContext();
com.netscape.server.IServerContext sc;


// legal cast within iAS
sc = (com.netscape.server.IServerContext) ctx;
com.kivasoft.IContext kivaContext = sc.getContext();
```

Example 3

As an alternative to Example 2, a caller can access the underlying AppLogic instance from a servlet and obtain the context there, as described in Example 1 for AppLogics. The servlet code would look like this:

```
HttpServletRequest req;
HttpServletRequest2 req2;
req2 = (HttpServletRequest2) req;  // legal cast within iPlanet Application Server
AppLogic al = req2.getAppLogic();
com.kivasoft.IContext kivaContext;
kivaContext = al.context;
```

Example 4

From a bean, the standard javax.ejb.SessionContext or javax.ejb.EntityContext can be cast to an IServerContext, and from there, a com.kivasoft.IContext instance can be obtained. In a bean, the code would look like this:

```
javax.ejb.SessionContext m_ctx;
.
.
.
com.netscape.server.IServerContext sc;
// legal cast within iPlanet Application Server
```

```
sc = (com.netscape.server.IServerContext) m_ctx;
com.kivasoft.IContext kivaContext;
kivaContext = sc.getContext();
```

Example 5

From a Java extension, an instance of the context is supplied to the init( ) method of your extension, as shown by the following code:

```
public int init(IObject obj) {
   com.kivasoft.IContext kivaContext = (com.kivasoft.IContext)
obj;
   .
   .
   .
```

Related Topics

com.kivasoft.applogic.AppLogic class (deprecated),
com.kivasoft.dlm.GXContext class,
com.netscape.server.ICallerContext Interface,
com.netscape.server.IServerContext Interface

IDataConn Interface *(deprecated)*

IDataConn is deprecated and is provided for backward compatibility only. New applications should use the java.sql.Connection interface from the JDBC Core API.

The IDataConn interface represents a connection to a relational data source.

IDataConn provides methods for preparing a query, executing a query, identifying table(s) to work with, and closing the connection explicitly. In addition, the data connection object is used in other operations for interacting with a data source.

IDataConn is part of the Data Access Engine (DAE) service. To create an instance of the IDataConn interface, use createDataConn( ) in the AppLogic class (deprecated), as shown in the following example:.

```
IDataConn conn;
if((conn = createDataConn(0, GX_DA_DAD_DRIVERS.GX_DA_DRIVER_ODBC,
"CATALOG", "CATALOG", "steve", "pass7878")) == null)
. . .
```

Package

com.kivasoft

Methods

Method	Description
closeConn( )	Explicitly closes a database connection.
createTrigger( )	Creates a new trigger object in the specified table.
disableTrigger( )	Disables a trigger associated with a specified table. This feature is supported by Oracle databases only.
dropTrigger( )	Removes a trigger from a specified table.
enableTrigger( )	Enables a trigger for a specified table. This feature is supported by Oracle databases only.
executeQuery( )	Executes a flat query on the data connection.
getConnInfo( )	Returns database and user information about the current database connection.
getConnProps( )	Returns registry information about the current database connection.
getDriver( )	Returns the identifier of the data source driver that the current database connection is using.
getTable( )	Returns the table definition object for the specified table.
getTables( )	Returns an IValList of database tables or views that are available to the specified user.
prepareCall( )	Creates an ICallableStmt object that contains a call to a stored procedure.
prepareQuery( )	Prepares a flat query object for subsequent execution.
setConnProps( )	Specifies registry values for the current database connection.

Related Topics

createDataConn( ) in the AppLogic class (deprecated)

getDataConn( ) in the ITable Interface (deprecated)

addRow( ), deleteRow( ), and updateRow( ) in the ITable Interface (deprecated)

closeConn( )

Explicitly closes the database connection.

Syntax

```
public int closeConn(
   int dwFlags)
```

dwFlags. Specify 0, or GX_DA_CLOSECONN_FLAGS.GX_DA_UNBIND_TRANS, which explicitly unbinds a physical connection from a transaction.

Usage

The Data Access Engine performs certain housekeeping tasks, such as shutdown and cleanup, automatically and intermittently. Use closeConn( ) to explicitly close a database connection and release system resources, such as when memory is low. Calling closeConn( ) breaks the virtual connection to the database and puts the physical connection back into the database connection cache.

Rules

- Closing the database connection changes the state of the IDataConn object to closed.
- Close a database connection only after the AppLogic no longer needs it. A run-time error will occur if subsequent operations attempt to use a data connection object that has already been closed.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
IDataConn conn;
if((conn = createDataConn(0, GX_DA_DAD_DRIVERS.GX_DA_DRIVER_ODBC,
"CATALOG", "CATALOG", "steve", "pass7878")) == null)
{
   // Cannot connect - return error message
   result("Failed to connect to Catalog db");
   return null;
};
// Connected - return the data connection object
// used for subsequent operations on the database
return conn;
};
. . . perform database operations . . .
```

```
// Explicitly close the connection
int closeResult;
closeResult=conn.closeConn(0);
```

Related Topics

createDataConn( ) in the AppLogic class (deprecated)

createTrigger( )

Creates a new trigger object in the specified table.

Syntax

```
int createTrigger(
   String pTable,
   String pName,
   String pCondition,
   String pOptions,
   String pSQLBlock);
```

pTable. The table on which the trigger is defined. You can specify the name of the owner as a prefix to the table name, for example, "jim.myTable".

pName. The name of the trigger object to create.

pCondition. The condition that determines whether or not the SQL procedure (defined in the pSQLBlock parameter) executes. For example, you can specify that the SQL procedure executes only if a column contains a specific value:

```
"FOR EACH ROW WHEN(city = 'San Francisco')"
```

pOptions. The row operations that determine when the trigger executes. For example, you can specify that the trigger be activated BEFORE or AFTER an INSERT, UPDATE, and/or DELETE operation:

```
"AFTER INSERT, UPDATE"
```

pSQLBlock. The definition of the SQL block to execute when the trigger goes into effect. Refer to your database documentation for information on the SQL block format.

Usage

A trigger is a SQL procedure associated with a table. It is automatically activated when a specified row operation, such as INSERT, UPDATE, and DELETE, is issued against the table. Use createTrigger( ) to specify the table and the data modification command that should activate the trigger, and the action or actions the trigger is to take.

Tips

- For specific information on supported trigger options and conditions, refer to the description of triggers in your database documentation.
- After creating a trigger, enable it by calling enableTrigger( ). The following are exceptions to the rule:
  - Sybase does not support the enabling or disabling of triggers.
  - Oracle automatically enables a trigger when the trigger is created; you can optionally call enableTrigger( ), but it will have no effect.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
IDataConn conn;
conn = createDataConn(0, GX_DA_DAD_DRIVERS.GX_DA_DRIVER_ODBC,
  "personnelDB", "personnelDB", "sandra", "pass7878");
conn.createTrigger("employees", "ProcessNew",
  "FOR EACH ROW WHEN(title='Director')",
  "AFTER INSERT", "[SQL instruction here]";
conn.enableTrigger("employees", "ProcessNew");
```

Related Topics

disableTrigger( )

dropTrigger( )

enableTrigger( )

disableTrigger( )

Disables a trigger associated with a specified table. This feature is supported by Oracle databases only.

Syntax

```
public int disableTrigger(
   String pTable,
   String pName);
```

pTable. The table in which the trigger is located.

pName. The name of the trigger to disable.

Usage

Use disableTrigger( ) to temporarily stop a trigger from being activated. The trigger is disabled until it is enabled with enableTrigger( ). To remove a trigger from a table permanently, use dropTrigger( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

createTrigger( )

dropTrigger( )

enableTrigger( )

dropTrigger( )

Removes a trigger from a specified table.

Syntax

```
public int dropTrigger(
   String pTable,
   String pName);
```

pTable. The table on which the trigger is defined.

pName. The name of the trigger to remove.

Usage

Use dropTrigger( ) to delete a trigger that is no longer required. To temporarily stop a trigger from being activated, use disableTrigger( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

createTrigger( )

disableTrigger( )

enableTrigger( )

enableTrigger( )

Enables a trigger for a specified table. This feature is supported by Oracle databases only.

Syntax

```
public int enableTrigger(
   String pTable,
   String pName);
```

pTable. The table on which the trigger is defined.

pName. The name of the trigger to enable.

Usage

Use enableTrigger( ) to prepare a specified trigger for activation. Call enableTrigger( ) after you create a trigger with createTrigger( ), or to enable a trigger that was disabled with disableTrigger( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

createTrigger( ); disableTrigger( )

dropTrigger( )

executeQuery( )

Executes a flat query on the data connection.

Syntax

```
public IResultSet executeQuery(
   int dwFlags,
   IQuery pQuery,
   ITrans pTrans,
   IValList pProps)
```

dwFlags. Specifies flags used to execute this query.

- For synchronous operations, the default, specify zero or GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_SYNC.
- For asynchronous operations, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC.
- To activate result set buffering, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_RS_BUFFERING.

The AppLogic can pass both result set buffering and either synchronous or asynchronous queries as the flags parameter, as shown in the following example:

```
(GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC |
GX_DA_EXECUTEQUERY_FLAGS.GX_DA_RS_BUFFERING)
```

pQuery. IQuery object that contains the flat query object to execute.

pTrans. ITrans object that contains the transaction to which this query applies, or null.

pProps. IValList object that contains query properties, or null for no properties. After instantiating an object of the IValList interface, set any of the following properties:

- RS_BUFFERING turns on result set buffering when set to “TRUE”.
- RS_INIT_ROWS specifies the initial size of the buffer, in number of rows. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED.
- RS_MAX_ROWS specifies the maximum number of rows for the buffer. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED.
- RS_MAX_SIZE specifies the maximum number of bytes for the buffer.

If RS_BUFFERING is enabled and if the optional parameters are not specified, the global values in the registry are used instead.

Rules

- Before calling executeQuery( ), AppLogic must create a query by first calling createQuery( ) in the AppLogic class (deprecated) to create the IQuery object, then using methods in the IQuery Interface (deprecated) to define the query.

- If the query is part of a transaction, before calling executeQuery( ), the AppLogic must first create the ITrans transaction object using createTrans( ) in the AppLogic class (deprecated), then begin the transaction using begin( ) in the ITrans Interface (deprecated), and then specify the ITrans object as a parameter when calling executeQuery( ).

Return Value

IResultSet object containing the result of the query, or null for failure (such as invalid tables or columns, or an invalid data comparison). If the result set is empty, calling getRowNumber( ) from the IResultSet Interface (deprecated) returns zero.

Example 1

```
// Create the flat query object
IQuery qry=createQuery();

// Set up the query
qry.setTables("CTLcust");
qry.setFields("CustomerID, Customer");
qry.setWhere("Customer"+"="+String.valueOf(custId));

// Execute the query
IResultSet rs=conn.executeQuery(0, qry, null, null);
// Check for a non empty result
if((rs!=null)&&(rs.getRowNumber()>0))
    return result("Sorry, this user ("+
    firstName+" "+lastName+") already exists");
// Otherwise, process the result set
```

Example 2

```
// Set up result set buffering for query
IValList props;
props = GX.CreateValList();
// Turn on result set buffering
props.setValString("RS_BUFFERING", "TRUE");
// Specify the maximum number of rows to buffer
props.setValInt("RS_MAX_ROWS", 50);
IQuery qry = createQuery();
. . . define query properties . . .
// Attempt to execute query with result set buffering
IResultSet rs = conn.executeQuery(0, qry, null, props);
```

Related Topics

createDataConn( ) in the AppLogic class (deprecated)

IQuery Interface (deprecated)

IResultSet Interface (deprecated)

ITrans Interface (deprecated)

IValList interface

getConnInfo( )

Returns database and user information about the current database connection.

Syntax

```
public IValList getConnInfo()
```

Usage

When the client code calls the createDataConn( ) method in the AppLogic class to create a connection between the client and the specified database, it passes the following parameters: flags, driver, datasource, database, username, and password. Once a data connection has been established, you can call getConnInfo( ) to return the datasource, database, user, and password values.

Tip

To return the driver value, use getDriver( ).

Return Value

IValList object, or null for failure.

getConnProps( )

Returns registry information about the current database connection.

Syntax

```
public IValList getConnProps()
```

Usage

Use getConnProps( ) to get database connection information that the iPlanet Application Server administrator set through the Enterprise Administrator. The information is returned in an IValList object that contains the following keys and values:

Key	Value
"cache_free_entries"	An integer indicating the number of slots set for free connections.
"cache_alloc_size"	An integer indicating the initial number of slots in the connection cache.
"conn_db_vendor"	A string that identifies the database vendor, for example, “Oracle” or “Sybase.”

The getConnProps( ) method might return other information depending on the database being used.

Applications typically use the database vendor information in conditional code that executes differently depending on the type of database.

Return Value
IValList object, or null for failure.

Related Topics
setConnProps( )

getDriver( )
Returns the identifier of the data source driver that the current database connection is using.

Syntax

Usage
When the client code calls the createDataConn( ) method in the AppLogic class to create a connection between the client and the specified database, it passes the following parameters: flags, driver, datasource, database, username, and password. Once a data connection has been established, you can call various methods in the IDataConn interface to return the values that were passed to createDataConn( ).

Call getDriver( ) to return the driver information.

Tip
To return the datasource, database, user, and password values, use getConnInfo( ).

Return Value

An int value corresponding to one of the static variables in the GX_DA_DAD_DRIVERS class.

GX_DA_DRIVER_ODBC	GX_DA_DRIVER_SYBASE_CTLIB
GX_DA_DRIVER_MICROSOFT_JET	GX_DA_DRIVER_MICROSOFT_SQL
GX_DA_DRIVER_INFORMIX_SQLNET	GX_DA_DRIVER_INFORMIX_CLI
GX_DA_DRIVER_INFORMIX_CORBA	GX_DA_DRIVER_DB2_CLI
GX_DA_DRIVER_ORACLE_OCI	GX_DA_DRIVER_DEFAULT

getTable( )

Returns the table definition object for the specified table.

Syntax

```
public ITable getTable(
   String szTable)
```

szTable. Name of the table to request. This can include the schema name, for example, “jim.myTable.” Do not use patterns or wildcards.

Usage

Use getTable( ) for the following reasons:

- To change data in the table using methods in the ITable Interface (deprecated) to insert, update, and delete rows.
- When the schema of a table is unknown, to obtain information about the table definition from the database catalog, such as table name, table columns, data connection, and so on.

Rule

The AppLogic usually calls getTable( ) only once to obtain a table definition. Subsequent calls return a separate ITable object that represents the same table. Each AppLogic can call getTable( ) and operate on its own copy of the table definition.

Tips

- If the table name is unknown, use getTables( ) to retrieve an IValList of tables in the data source, then use methods in the IValList interface and the Val class to iterate through the table names obtained and determine which table to retrieve.
- To obtain additional information about individual columns, use the IColumn Interface (deprecated).

Return Value

ITable object, or null for failure (such as an invalid table name).

Example

```
ITable table = dbConn.getTable("CTLcust");
if(table==null) return result("Can't find the table:
"+"CTLcust");
// Otherwise, process the table columns
int cCustId = table.getColumnOrdinal("CustomerID");
int cFirst = table.getColumnOrdinal("FirstName");
. . .
```

Related Topics

ITable Interface (deprecated)

createDataConn( ) in the AppLogic class (deprecated)

Val class

IValList interface

getTables( )

Returns an IValList of database tables or views that are available to the specified user.

Syntax

```
public IValList getTables(
   String szQualifier,
   String szOwner,
   String szTable)
```

szQualifier. Specify null. Driver-dependent.

szOwner. Specify null, or a schema name, which returns tables for that schema.

szTable. Table or view name with wildcards, or null for all tables. Wildcards must be in the format supported by the data source. For example, you can use search patterns using the following characters:

- underscore (_) for single characters
- percent sign (%) for any sequence of zero or more characters

Usage

Use getTables( ) when the list of available tables on the data source is unknown. The AppLogic can obtain a subset of available tables by specifying wildcards in the table name.

Rules

- The AppLogic must be logged in with sufficient privileges to obtain a list of tables from the database. For more information, see your database server documentation.
- The AppLogic must specify a valid table name, view name, or name pattern. Aliases and synonyms are not supported for security reasons.

Tip

Use methods in the IValList interface and the Val class to iterate through the table names obtained and determine which table(s) to work with. Thereafter, use to access each table.

Return Value

IValList object containing a list of table names, or null for failure (such as if none are found to match the search expression).

Related Topics

ITable Interface (deprecated)

createDataConn( ) in the AppLogic class (deprecated)

IValList interface

prepareCall( )

Creates an ICallableStmt object that contains a call to a stored procedure.

Syntax

```
public ICallableStmt prepareCall(
   int dwFlags,
   IQuery pQuery,
   ITrans pTrans,
   IValList pProps);
```

dwFlags. Specify 0.

pQuery. The IQuery object that contains the call to a stored procedure. The stored procedure call should have been specified with the setSQL( ) method in the IQuery interface.

pTrans. ITrans object that contains the transaction associated with this callable statement, or null for no transaction. This same ITrans object must then be passed to the execute( ) method of the ICallableStmt interface.

pProps. Specify null.

Usage

Use prepareCall( ) to create a ICallableStmt object that contains a call to a stored procedure. After creating the callable statement, run it by calling execute( ) in the ICallableStmt interface.

Rules

- Before calling prepareCall( ), the AppLogic must create a query by first calling createQuery( ) in the AppLogic class (deprecated) to create the IQuery object, then using the setSQL( ) method in the IQuery Interface (deprecated) to define the call to a stored procedure.
- When accessing a stored procedure on Sybase or MS SQL Server, input parameter names must be prefixed with the ampersand (&) character, for example, &param1. Other database drivers accept the ampersand, as well as, the colon (:) character. For all database drivers, input/output and output parameter names are prefixed with the colon (:) character, for example, :param2.

Example

```
// Create the database connection.
IDataConn conn_rtest = createDataConn(0,
GX_DA_DAD_DRIVERS.GX_DA_DRIVER_DEFAULT,
/* Datasource name */ "ksample",
/* Database name   */ "",
/* userName        */ "kdemo",
```

```
/* password        */ "kdemo");

//Write command to call stored procedure
String theProc = "{call myProc1(&p1, :p2)}";
IQuery myquery;
myquery = createQuery();
myquery.setSQL(theProc);

//Prepare the callable statement for execution
ICallableStmt myStmt;
myStmt = conn_rtest.prepareCall(0, myquery, null, null);

// Set parameters and run the callable statement
```

Return Value

ICallableStmt object, or null for failure.

Related Topics

ICallableStmt Interface (deprecated)

prepareQuery( )

Prepares a flat query object for subsequent execution.

Syntax

```
public IPreparedQuery prepareQuery(
   int dwFlags,
   IQuery pQuery,
   ITrans pTrans,
   IValList pProps)
```

dwFlags. Specify 0.

pQuery. IQuery object that contains the query or statement to execute.

pTrans. ITrans object that contains the transaction to which this query applies, or null. This same object must then be passed to the execute( ) method of the IPreparedQuery interface.

pProps. Specify null.

Usage

Use prepareQuery( ) to prepare the query, then execute the prepared query using execute( ) in the IPreparedQuery Interface (deprecated). An application can also use prepareQuery( ) with result set buffering to pre-fetch result set data efficiently from a back-end database.

Rule

Before calling prepareQuery( ), AppLogic must create a query by first calling createQuery( ) in the AppLogic class (deprecated) to create the IQuery object, then using methods in the IQuery Interface (deprecated) to define the query.

Return Value

IPreparedQuery object, or null for failure.

Example

```
// Create the data connection
IDataConn conn;
conn = createDataConn(0, GX_DA_DAD_DRIVERS.GX_DA_DRIVER_DEFAULT,
"Orders", "Orders", "user", "password");

// Create the flat query and prepared query objects
IQuery qry = createQuery();
IPreparedQuery pqry;
IValList params;
params = GX.CreateValList();

// Set up the INSERT statement
qry.setSQL("INSERT INTO TABLE Products (ProductName,
QuantityPerUnit) VALUES (:name, :quant)");

// Prepare the flat query
pqry = conn.prepareQuery(0, qry, null, null);

// Specify a set of query parameters, then execute
params.setValString(":name","Chicken Dumplings");
params.setValString(":quant", "48 packages");
IResultSet rs1 = pqry.execute(0, params, null, null);
. . . process rs1. . .

// Specify different set of query parameters, then execute
params.setValString(":name", "Rice Noodles");
params.setValString(":quant", "96 packages");
IResultSet rs2 = pqry.execute(0, params, null, null);
. . . process rs2. . .
```

Related Topics

IPreparedQuery Interface (deprecated)

IQuery Interface (deprecated)

ITrans Interface (deprecated)

IValList interface

createDataConn( ) in the AppLogic class (deprecated)

setConnProps( )

Specifies registry values for the current database connection.

Syntax

```
public int setConnProps(
    IValList pProps)
```

pProps. A pointer to the IValList object that contains the connection properties to set in the registry. Use the following defined key names for the connection properties:

Key	Value
"cache_free_entries"	An integer indicating the number of slots set for free connections.
"cache_alloc_size"	An integer indicating the initial number of slots in the connection cache.

Usage

Use setConnProps( ) to override database connection properties that the iPlanet Application Server administrator set through the Enterprise Administrator. To get the current connection properties programmatically, call getConnProps( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

getConnProps( )

IDataConnSet Interface *(deprecated)*

IDataConnSet is deprecated and is provided for backward compatibility only. New applications should use the JDBC API to provide similar functionality.

The IDataConnSet interface represents a collection of data connections and associated query names. It is used in conjunction with loading a query file.

Use IDataConnSet when loading a hierarchical query from a file. The AppLogic first establishes a data connection with each database on which any queries will be run. Next, the AppLogic calls createDataConnSet( ) in the AppLogic class (deprecated) to create an empty IDataConnSet object, then populates this object with query name / data connection pairs.

In this way, the AppLogic can use parameterized queries and select and assign data connections dynamically at runtime. Finally, the AppLogic calls loadHierQuery( ) in the AppLogic class (deprecated) to create the hierarchical query object.

IDataConnSet is part of the Data Access Engine (DAE) service.

To create an instance of the IDataConnSet interface, use createDataConnSet( ) in the AppLogic class (deprecated), as shown in the following example:.

```
IDataConnSet connSet;

connSet = createDataConnSet();
```

Package

com.kivasoft

Methods

addConn( )	Associates a query name with a data connection object and adds it to the IDataConnSet object.

Related Topics

createDataConnSet( ) in the AppLogic class (deprecated)

addConn( )

Associates a query name with a data connection object and adds it to the IDataConnSet object.

Syntax

```
public int addConn(
    String pQueryName,
    IDataConn pConn)
```

pQueryName. Name of a query in the query file.

pConn. Name of the data connection object representing an active connection with the data source on which the query will be run.

Rules

- Every named query in the qulery file must have a corresponding named query in the IDataConnSet object.
- The AppLogic must first create the data connection object using createDataConn( ) in the AppLogic class (deprecated).
- Duplicate query names are not permitted.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
IDataConnSet connSet;
connSet = createDataConnSet();
// Specify query / db connection pairs
connSet.addconn("employee", conn_empDB);
connSet.addconn("sales", conn_salesDB);
IHierQuery hqry;
// Load the GXQ file with the db connection set
hqry = loadHierQuery("employeeReport.gxq",connSet, 0, null);
// Run the report
evalTemplate("employeeReport.html", hqry);
```

Related Topics

createDataConnSet( ) in the AppLogic class (deprecated)

IEnumObject Interface

The IEnumObject interface represents an enumeration object that contains IObject instances. Some methods that return a list of objects, such as enumEvents( ) in the IAppEventMgr interface, return an IEnumObject object.

The IEnumObject interface defines methods for counting and accessing the IObject instances in an IEnumObject.

Package

com.kivasoft

Methods

Method	Description
enumCount( )	Returns the number of IObject instances in an IEnumObject.
enumNext( )	Returns the next IObject instance in an IEnumObject.
enumReset( )	Resets to the first IObject instance in an IEnumObject.

Related Topics

enumEvents( ) in the IAppEventMgr Interface

enumCount( )

Returns the number of IObject instances in an IEnumObject.

Syntax

```
public int enumCount()
```

Usage

Use enumCount( ) to determine the number of objects to process before iterating through the IObject instances in the IEnumObject.

Return Value

The number of objects in an IEnumObject.

Example

In the following example, enumEvents( ) returns all the application events registered with the iPlanet Application Server in an IEnumObject. The enumCount( ) method is used in conjunction with enumNext( ) and enumReset( ) to access objects in the IEnumObject.

```
// suppose appEventMgr holds a valid reference to
//     an IAppEventMgr instance

// Get the Enumeration object for all registered appevents
IEnumObject enumObj = appEventMgr.enumEvents();

// Retrieve the count of registered appevents
int count = enumObj.enumCount();
try {
    p.writeObject("Number of Registered Events: ");
    p.writeInt(count);
} catch (IOException e) {
    return streamResult("Failed to write to report file<br>");
}

// Reset to the first object in the enumeration object
enumObj.enumReset(0);

// Iterate through all the enumeration instances
while (count > 0) {

// Process the objects.
}
```

Related Topics

enumEvents( ) in the IAppEventMgr Interface

enumNext( )

enumReset( )

enumNext( )

Returns the next IObject instance in an IEnumObject.

Syntax

```
public IObject enumNext()
```

Usage

Use enumNext( ) in conjunction with enumCount( ) and enumReset( ) to iterate through an IEnumObject.

Return Value

GXE.SUCCESS if the method succeeds.

Example

In the following example, enumEvents( ) returns all the application events registered with the iPlanet Application Server in an IEnumObject. The enumNext( ) method is used in conjunction with enumCount( ) and enumReset( ) to access objects in the IEnumObject.

```
// Suppose appEventMgr holds a valid reference to AppEvent
Manager

// Get the Enumeration object for all registered appevents
IEnumObject enumObj = appEventMgr.enumEvents();

// Retrieve the count of registered appevents
int count = enumObj.enumCount();
try {
    p.writeObject("Number of Registered Events: ");
    p.writeInt(count);
} catch (IOException e) {
    return streamResult("Failed to write to report file<br>");
}

// Reset to the first object in the enumeration object
enumObj.enumReset(0);

// Iterate through all the enumeration instances
while (count > 0) {
    IObject vListObj = enumObj.enumNext();
    if (vListObj instanceof IAppEventObj) {
        IAppEventObj vAppEventObj = (IAppEventObj)vListObj;

        // process the appevent
    }
}
```

Related Topics

enumEvents( ) in the IAppEventMgr Interface

enumCount( )

enumReset( )

enumReset( )

Resets to the first IObject instance in an IEnumObject.

Syntax

```
public int enumReset(
   int dwFlags)
```

dwFlags. Specify 0.

Usage

Use enumReset( ) before iterating through an IEnumObject. Doing so ensures that iteration begins at the first IObject instance in the IEnumObject.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Retrieve the count of objects in the IEnumObject
int count = enumObj.enumCount();

// Reset to the first object in the enumeration object
enumObj.enumReset(0);

// Iterate through all the enumeration instances
while (count > 0) {
   IObject vListObj = enumObj.enumNext();
   // Process the objects
```

Related Topics

enumEvents( ) in the IAppEventMgr Interface

enumCount( )

enumNext( )

IError Interface *(deprecated)*

IError is deprecated and is provided for backward compatibility only. New applications should use JDBC exceptions instead. For example, the java.sql package provides exceptions such as BatchUpdateException, DataTruncation, SQLException, and SQLWarning.

The IError interface represents an error code object that consists of a code and a corresponding error message that originates from a facility, such as an operating system or a database. In this release, IError handles database errors only.

Use the methods in the IError interface to get error codes and messages returned by a database.

The IErrorinterface is implemented by the IDataConn object. To use it, cast IDataConn to the IError interface, as shown in the following example:

```
IDataConn conn;
conn = createDataConn(...);
IError error;
error = (IError) conn;
```

Package

com.kivasoft

Methods

Method	Description
getErrorCode( )	Returns the current error code as a string.
getErrorCodeNum( )	Returns the current error code as a number.
getErrorMessage( )	Returns the message associated with the current error code.
getErrorFacility( )	Returns a description of the facility that generated an error code.

getErrorCode( )

Returns the current error code as a string.

Syntax

```
public String getErrorCode()
```

Usage

Use getErrorCode( ) after a database operation, such as running a stored procedure or executing a query, to retrieve the error code for debugging or error-handling purposes. The following is an example of a returned error code: "ORA-03130".

Tip

For ODBC, the error codes usually consist of the ODBC error code and the database error code separated by a space, for example, "S1000 1017". Sometimes just the ODBC error code, such as "S1000", is returned.

Return Value

The current error code as a string.

Related Topics

getErrorCodeNum( )

getErrorMessage( )

getErrorFacility( )

getErrorCodeNum( )

Returns the current error code as a number.

Syntax

```
public int getErrorCodeNum()
```

Usage

Use getErrorCodeNum( ) after a database operation, such as running a stored procedure or executing a query, to retrieve the error code for debugging or error-handling purposes.

Return Value

The current error code as a number. If the error code cannot be converted to a number, getErrorCodeNum( ) returns null. For example, an error code that contains letters, such as "S1000", cannot be converted to a number.

Related Topics

getErrorCode( )

getErrorMessage( )

getErrorFacility( )

getErrorMessage( )

Returns the message associated with the current error code.

Syntax

```
public String getErrorMessage(
   String pCode)
```

pCode. Specify any string.

Usage

Use getErrorMessage( ) after a database operation, such as running a stored procedure or executing a query, to retrieve the message associated with the current error code. The AppLogic can then display the message to users. The following is an example of a returned error message: "[ODBC][Visigenic driver][S1000]Connection attempt failed".

Return Value

The message associated with the current error code.

Related Topics

getErrorCode( )

getErrorCodeNum( )

getErrorFacility( )

getErrorFacility( )

Returns a description of the facility that generated an error code.

Syntax

```
public String getErrorFacility()
```

Usage

Use getErrorFacility( ) after a database operation, such as running a stored procedure or executing a query, to get information on which driver generated the current error code. The following is an example of a description returned by getErrorFacility( ): "ODBC DAD".

Return Value

The description of the facility that generated the current error code.

Related Topics

getErrorCode( )

getErrorCodeNum( )

getErrorMessage( )

IHierQuery Interface *(deprecated)*

IHierQuery is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model.

For information about replacing IHierQuery functionality in existing applications, see the *Migration Guide*.

The IHierQuery interface represents a hierarchical query. IHierQuery provides methods for retrieving hierarchical information organized in nested levels of detail, as in the following example:

```
Asia          170
  China       110
  Japan        60
Europe         80
  France       70
  Portugal     10
```

A hierarchical query combines multiple flat queries organized in cascading, parent-child relationships. Each query is an IQuery object containing data selection criteria. The IHierQuery object contains the definition of the hierarchical structure of parent-child relationships among IQuery objects.

To use a hierarchical query, the AppLogic first creates each individual query and defines its selection criteria. Next, it creates theIHierQueryobject and calls addQuery( ) repeatedly to add a child query to a parent query for each level of detail in the hierarchical query.

After the hierarchical query is constructed, the AppLogic calls its execute( ) method to run the hierarchical query on the target data source and retrieve a hierarchical result set in an IHierResultSet object.

Alternatively, the AppLogic can load a hierarchical query stored in a file. For more information, see loadHierQuery( ) and createDataConnSet( ) in the AppLogic class (deprecated).

To create an instance of the IHierQuery interface, use createHierQuery( ) in the AppLogic class (deprecated), as shown in the following example:

```
IHierQuery hqry;
hqry = createHierQuery();
```

Package

com.kivasoft

Methods

Method	Description
addQuery( )	Adds a child query to a parent query, defining an additional level of detail in the hierarchical query.
delQuery( )	Removes a child query from its parent query.
execute( )	Executes a hierarchical query and returns a hierarchical result set.

Example 1

```
// Create the flat query
IQuery qry = createQuery();
qry.setTables("CTLusers");
qry.setFields("loginName, Password, AccessLevel")
qry.setOrderBy("LoginName);

// Create the hierarchical query used for template processing
IHierQuery hqry = createHierQuery();

// Add the flat query object and data connection to hqry
hqry.addQuery(qry, conn, "USERS", "", "");

// Pass hierarchical query to evalTemplate() for reporting
if(evalTemplate("apps/template/userinfo.html", hqry)==0)
   return result("");
else
   return result("Failed to Generate HTML");
}
```

Example 2

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

addQuery( )

Adds a child query to a parent query, defining an additional level of detail in the hierarchical query.

Syntax

```
public int addQuery(
    IQuery pQuery,
    IDataConn pConn,
    String szAlias,
    String szParent,
    String szJoin)
```

pQuery. IQuery object that contains the flat query object to append as a child to the parent query.

pConn. IDataConn object that contains the data connection where the child query will be executed. Each flat query in the hierarchical query can retrieve data from a different data source.

szAlias. Name used to uniquely identify this child query in the query hierarchy. AppLogic must specify a child name that is unique within the hierarchical query.

szParent. Name of the parent query to contain this child query. Use an empty string ("") for the highest level in the hierarchical query. When adding a child query to an existing parent query, the specified parent name must have already been specified in a previous addQuery( ) call.

szJoin. Join clause used to specify a join for this query, defining the relationship between a field in the child query and a field in the parent query. Use an empty string for the highest level in the hierarchical query. Use the following Netscape-compliant syntax for the join clause:

```
"ParentQuery.table.column='childQuery.table.column'"
```

Optionally, you can specify the schema:

```
"ParentQuery.schema.table.column='childQuery.schema.table.column'"
```

NOTE The only difference between the Netscape Application Server and SQL join syntax is that, with Netscape, you prepend the clause with the query name.

To refer to a field name in the parent query, include the parent query name before the field name, as shown in the following example, in which CITY is the name of the parent query:

```
hqr.addQuery(qryEMP, conn, "EMP", "CITY", "EMP.employee.city =
'CITY.city'");
```

Use the AND and OR operators to specify additional join conditions. Use parentheses to specify the order of precedence in complex join criteria.

Usage

Use addQuery( ) when constructing the hierarchical query to define the hierarchical relationships among child and parent queries. The number of nested levels, and thus the number of addQuery( ) calls, is limited only by system resources.

Rules

- The AppLogic must first create the data connection using createDataConn( ) in the AppLogic class (deprecated).
- The AppLogic must then create the specified child query using createQuery( ) in the AppLogic class (deprecated). A separate child query must exist for every level of data.

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Create the flat query
IQuery qry = createQuery();
qry.setTables("CTLusers");
qry.setFields("loginName, Password, AccessLevel")
qry.setOrderBy("LoginName);

// Create the hierarchical query used for template processing
IHierQuery hqry = createHierQuery();

// Add the flat query object and data connection to hqry
hqry.addQuery(qry, conn, "USERS", "", "");
```

```
// Pass hierarchical query to evalTemplate() for reporting
if(evalTemplate("apps/template/userinfo.html", hqry)==0)
   return result("");
else
   return result("Failed to Generate HTML");
}
```

Example 2

```
// Create the City flat query
IQuery qryCTY;
qryCTY = createQuery();
qryCTY.setTables("emp");
qryCTY.setFields("city, Sum(salary) as sumsalary");
qryCTY.setGroupBy("city");

// Create the Employee flat query
IQuery qryEMP;
qryEMP = createQuery();
qryEMP.setTables("emp");
qryEMP.setFields("name, salary, city");

IHierQuery hqry;
hqry = createHierQuery();
// Add both queries to the hierarchical query object
hqry.addQuery(qryCTY, conn, "CTY", "", "");
hqry.addQuery(qryEMP, conn, "EMP", "CTY", "EMP.emp.city =
'CTY.city'");
// Run the report
evalTemplate("GXApp/EmpTrack/Templates/report.html", hqry);
return result(null);
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IQuery Interface (deprecated)

IDataConn Interface (deprecated)

delQuery( )

Removes a child query from its parent query.

Syntax

```
public int delQuery(
   String szName)
```

szName. Name of the child query to remove.

Usage

Use delQuery( ) to remove a child query that is no longer needed. Any children of the deleted child query are also removed.

Rule

The specified child query must exist in the hierarchical query.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Create the City flat query
IQuery qryCTY;
qryCTY = createQuery();
qryCTY.setTables("emp");
qryCTY.setFields("city, Sum(salary) as sumsalary");
qryCTY.setGroupBy("city");

// Create the Employee flat query
IQuery qryEMP;
qryEMP = createQuery();
qryEMP.setTables("emp");
qryEMP.setFields("name, salary, city");

IHierQuery hqry;
hqry = createHierQuery();
// Add both queries to the hierarchical query object
hqry.addQuery(qryCTY, conn, "CTY", "", "");
hqry.addQuery(qryEMP, conn, "EMP", "CTY", "EMP.emp.city =
'CTY.city'");

// Execute the hierarchical query
IHierResultSet hrs = hqry.execute(0, 0, null);
if(hrs.getRowNumber("USERS")!=0)
   // Process rows in result set . . .

// Remove child query from hierarchical query
hqry.delQuery("qryEMP");

// Pass parent query to evalTemplate() for reporting
if(evalTemplate("apps/template/ctySumm.html", hqry)==0)
   return result("");
```

```
else
   return result("Failed to Generate HTML");
}
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

loadHierQuery( ) in the AppLogic class (deprecated)

execute( )

Executes a hierarchical query and returns a hierarchical result set.

Syntax

```
public IHierResultSet execute(
   int dwFlags,
   int dwTimeout,
   IValList pProps)
```

dwFlags. Specifies flags used to execute this hierarchical query.

- For synchronous operations, the default, specify zero or GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_SYNC.
- For asynchronous operations, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC.
- To activate result set buffering, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_RS_BUFFERING.

The AppLogic can pass both result set buffering and either synchronous or asynchronous queries as the flags parameter, as shown in the following example:

(GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC | GX_DA_EXECUTEQUERY_FLAGS.GX_DA_RS_BUFFERING).

dwTimeout. Specify 0 (zero).

pProps. IValList object that contains query properties, or null for no properties. After instantiating an object of the IValList interface, set any of the following properties:

- RS_BUFFERING turns on result set buffering when set to “TRUE”.

- RS_INIT_ROWS specifies the initial size of the buffer, in number of rows. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED.
- RS_MAX_ROWS specifies the maximum number of rows for the buffer. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED.
- RS_MAX_SIZE specifies the maximum number of bytes for the buffer.

If RS_BUFFERING is enabled and if the optional parameters are not specified, the global values in the registry are used instead..

Usage

After constructing a hierarchical query using addQuery( ), the AppLogic uses execute( ) to execute the query on the database server. Results are returned in a hierarchical result set.

Rule

AppLogic must first construct the hierarchical query using addQuery( ).

Return Value

IHierResultSet object that contains the result set of the hierarchical query, or null for failure (such as invalid parameters).

Example

```
// Create the flat query
IQuery qry = createQuery();
qry.setTables("CTLusers");
qry.setFields("loginName, Password, AccessLevel")
query.setWhere("UserId"+"="+wantedUser.hashCode());

// Create the hierarchical query
IHierQuery hqry = createHierQuery();
hqry.addQuery(qry, conn, "USERS", "", "");

// Execute the hierarchical query
IHierResultSet hrs = hqry.execute(0, 0, null);

// Process rows in result set
if(hrs.getRowNumber("USERS")>0)
. . .
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

loadHierQuery( ) in the AppLogic class (deprecated)

IHierResultSet Interface (deprecated)

IValList interface

IHierResultSet Interface *(deprecated)*

IHierResultSet is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model.

For information about replacing IHierResultSet functionality in existing applications, see the *Migration Guide*.

The IHierResultSet interface represents a hierarchical result set retrieved by a hierarchical query. IHierResultSet provides methods to iterate through rows in the hierarchical result set and retrieve information about each row. Alternatively, an AppLogic can process hierarchical result sets by passing them directly to the Template Engine using evalOutput( ) in the AppLogic class (deprecated).

IHierResultSet is part of the Data Processing Engine (DPE) service. To create an instance of IHierResultSet, use execute( ) in the IHierQuery Interface (deprecated), as shown in the following example:

```
IHierResultSet hrs
hrs = hqry.execute(0, 0, null);
```

Package

com.kivasoft

Methods

Method	Description
count( )	Returns the total number of rows retrieved so far from the data source for the specified child query.
getColumn( )	Returns the column definition for the column with the specified name in the specified child query.
getColumnByOrd( )	Returns the column definition for the column in the specified ordinal position for the specified child query.

Method	Description
getResultSet( )	Returns the result set for a specified child query.
getRowNumber( )	Returns the number of the current row for the specified child query in the hierarchical result set.
getValueDateString( )	Returns the value of a Date type column, as a string, from the specified child query in the result set.
getValueDouble( )	Returns the value of a double type column from the specified child query in the result set.
getValueInt( )	Returns the value of an int type column from the specified child query in the result set.
getValueString( )	Returns the value of a string type column from the specified child query in the result set.
moveNext( )	Moves to the next row for the specified child query in the result set.
moveTo( )	Moves to the specified row for the specified child query in the result set.

Example

The following code runs a hierarchical query and with the returned hierarchical result set, checks a user's access level to determine which listbox options to display:

```
// Create the hierarchical query
IHierQuery hqry = createHierQuery();
hqry.addQuery(qry, conn, "USERS", "", "");

// Execute the query
IHierResultSet hrs = hqry.execute(0, 0, null);

TemplateMapBasic map = new TemplateMapBasic();

// Determine whether result returned match
if(hrs.getRowNumber("USERS")==0)
   return result("No user matches loginname: "+wantedUser);
String access=hrs.getValueString("USERS", "AccessLevel");
String selAdmin, selNormal;
if(access.compareTo("AccessAdmin")==0)
{
   selAdmin="<option selected>"+"AccessAdmin"+"</option>\n";
   selNormal="<option>Normal</option>\n";
}
else
```

```
{
   selAdmin="<option>"+"AccessAdmin"+"</option>\n";
   selNormal="<option selected>Normal</option>\n";
}
String select="<select
name=accessControlLevel>\n"+selNormal+selAdmin+"</select>";

TemplateMapBasic tMap=new TemplateMapBasic();
tMap.put("ACCESS", select);

int ers = evalTemplate(template, (ITemplateData) hrs, tMap);
if (ers==0)
   return result("");
else
   return result("Failed to Generate HTML");
}
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IHierQuery Interface (deprecated)

count( )

Returns the total number of rows retrieved so far from the data source for the specified child query.

Syntax

```
public int count(
   String qryName)
```

qryName. Name of the child query that generated the result set.

Usage

Use count( ) to return the current number of rows processed so far in the result set. If iterating through rows in a result set that has been completely returned, use count( ) to determine the current maximum number of rows to process.

Tip

If result set buffering is enabled, the AppLogic can use count( ) to find the current number of rows in the buffer.

Rule

The specified child query must exist in the result set.

Return Value

An int value representing the number of rows in the result set retrieved so far, or zero for failure (such as an invalid child query name).

Example

```
// Create the hierarchical query
IHierQuery hqry = createHierQuery();
hqry.addQuery(qry, conn, "USERS", "", "");
// Execute the query
IHierResultSet hrs = hqry.execute(0, 0, null);
// Determine whether result returned match
if(hrs.count("USERS")==0)
    return result("No user matches loginname: "+wantedUser);
String access=hrs.getValueString("USERS", "AccessLevel");
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IHierQuery Interface (deprecated)

getColumn( )

Returns the column definition for the column with the specified name in the specified child query.

Syntax

```
public IColumn getColumn(
   String qryName,
   String colName)
```

qryName. Name of the child query that generated the result set.

colName. Name of the column. Must *not* be qualified with the schema name or table name (if necessary, use column alias to ensure that the colName is unambiguous).

Usage

Use getColumn( ) when the data definition of the column is unknown and is required for subsequent operations. The AppLogic can then use methods in the IColumn Interface (deprecated) to obtain descriptive information about a table column from the database catalog, such as the column name, precision, scale, size, table, and data type.

Rules

- The specified child query must exist in the result set.
- The specified column name must exist in the result set.

Tips

- Use getColumnByOrd( ) instead when the column position is known but its name is unknown.
- Columns that are the result of query expressions or formulas, such as `invoice.count * product.price`, should have a column alias in the result set. AppLogic can call setFields( ) in the IQuery Interface (deprecated) to specify field aliases using the "as" keyword.

Return Value

IColumn object containing information from the retrieved column, or null for failure (such as an invalid child query name or column name).

Example

```
// Obtain column information for UserID field
IColumn col;
col = hrs.getColumn("USERS", "UserID")
. . . process column info using IColumn methods . . .
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IHierQuery Interface (deprecated)

IColumn Interface (deprecated)

getColumnByOrd( )

Returns the column definition for the column in the specified ordinal position for the specified child query.

Syntax

```
public IColumn getColumnByOrd(
   String qryName,
   int colIndex)
```

qryName. Name of the child query that generated the result set.

colIndex. Ordinal position of a column in the result set. The ordinal position of the first column in the result set is 1, the second column is 2, and so on.

Usage

Use getColumnByOrd( ) when the data definition of the column is unknown and is required for subsequent operations. AppLogic can then use methods in the IColumn Interface (deprecated) to obtain descriptive information about a table column from the database catalog, such as the column name, precision, scale, size, table, and data type.

Rules

- The specified child query must exist in the result set.
- The specified column position must exist in the result set.

Tip

Use getColumn( ) instead when the column name is known but its ordinal position is unknown.

Return Value

IColumn object containing information from the retrieved column, or null for failure (such as an invalid child query name or column name).

Example

```
// Obtain column information for UserID field

IColumn col1, col2, col3, col4;

col1 = hrs.getColumnByOrd("USERS", 1)

col2 = hrs.getColumnByOrd("USERS", 2)

col3 = hrs.getColumnByOrd("USERS", 3)

col4 = hrs.getColumnByOrd("USERS", 4)

. . . process column info using IColumn methods . . .
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IHierQuery Interface (deprecated)

IColumn Interface (deprecated)

getResultSet( )

Returns the result set for a specified child query.

Syntax

```
public IResultSet getResultSet(
   String qryName)
```

qryName. Name of the child query that generated the result set to retrieve.

Usage

Use getResultSet( ) to retrieve and manipulate a particular child result set in the hierarchical result set. The AppLogic can then use methods in the IResultSet Interface (deprecated) to get data from the result set columns.

Rule

The specified child query must exist in the result set.

Return Value

IResultSet object , or null for failure (such as an invalid child query name).

Example

```
// Look up list of customers matching criteria from database
IHierResultSet hrs = lookupCustomer(ssn, lastName, firstName,
acctNum);


if (hrs == null)
{
   return handleOBSystemError("Could not retrieve database
results");
}
// Check the result set to see if any customers are found
IResultSet rs = hrs.getResultSet("SelCusts");
```

getRowNumber( )

Returns the number of the current row for the specified child query in the hierarchical result set.

Syntax

```
public int getRowNumber(
   String qryName)
```

qryName. Name of the child query that generated the result set.

Usage

When iterating through rows in a child set, use getRowNumber( ) to keep track of the number of rows processed.

Rule

The specified child query must exist in the result set.

Return Value

An int value representing the current row number in the child result set, or zero if the result set is empty or for failure (such as an invalid row or query name). The number of the first row in the result set is 1, the second row is 2, and so on. If zero is returned the first time AppLogic calls getRowNumber( ), that means the result set is empty for the specified child query.

Example

```
// Create the hierarchical query
IHierQuery hqry = createHierQuery();


hqry.addQuery(qry, conn, "USERS", "", "");


// Execute the query
IHierResultSet hrs = hqry.execute(0, 0, null);


TemplateMapBasic map = new TemplateMapBasic();
// Determine whether result returned match


if(hrs.getRowNumber("USERS")==0)
   return result("No user matches loginname: "+wantedUser);
String access=rs.getValueString("USERS", "AccessLevel");
String selAdmin, selNormal;
if(access.compareTo("AccessAdmin")==0)
{
   selAdmin="<option selected>"+"AccessAdmin"+"</option>\n";
   selNormal="<option>Normal</option>\n";
```

```
}
else
{
   selAdmin="<option>"+"AccessAdmin"+"</option>\n";
   selNormal="<option selected>Normal</option>\n";
}
String select="<select
name=accessControlLevel>\n"+selNormal+selAdmin+"</select>";
TemplateMapBasic tMap=new TemplateMapBasic();
tMap.put("ACCESS", select);
int ers = evalTemplate(template, (ITemplateData) rs, tMap);
if (ers==0)
   return result("");
else
   return result("Failed to Generate HTML");
}
```

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IHierQuery Interface (deprecated)

getValueDateString( )

Returns the value of a Date type column, as a string, from the specified child query in the result set.

Syntax

```
public String getValueDateString(
   String qryName,
   String colIName)
```

qryName. Name of the child query that generated the result set.

colName. Name of the column from which to retrieve the date.

Usage
Use getValueDateString( ) to retrieve date values from the result set for subsequent processing. The following is an example of the format in which getValueDateString( ) returns a date:

```
Jan 26 1998 12:35:00
```

Rule
The specified column must be a Date, Date Time, or Time data type.

Return Value
The date value as a string, or null for failure (such as an invalid column number or data type mismatch).

Related Topics
getValueDouble( )

getValueInt( )

getValueString( )

getValueDouble( )
Returns the value of a double type column from the specified child query in the result set.

Syntax
```
public double getValueDouble(
   String qryName,
   String colName)
```

qryname. Name of the child query that generated the result set.

colName. Name of the column from which to retrieve the double value.

Usage
Use getValueDouble( ) to retrieve decimal, floats, real, numeric, and double values from the result set for subsequent processing.

Rule
The specified column must be a double data type.

Return Value
A double value, or zero for failure (such as an invalid column number or data type mismatch).

Related Topics
getValueDateString( )

getValueInt( )

getValueString( )

getValueInt( )

Returns the value of an int type column from the specified child query in the result set.

Syntax
```
public int getValueInt(
   String qryName,
   String colName)
```

qryname. Name of the child query that generated the result set.

colName. Name of the column from which to retrieve the value.

Usage
Use getValueInt( ) to retrieve int or long values from the result set for subsequent processing.

Rule
The specified column must be an int or long data type.

Return Value
An int value, or zero for failure (such as an invalid column number or data type mismatch).

Related Topics
getValueDateString( )

getValueDouble( )

getValueString( )

getValueString( )

Returns the value of a string type column from the specified child query in the result set.

Syntax

```
public String getValueString(
   String qryName,
   String colName)
```

qryname. Name of the child query that generated the result set.

colName. Name of the column from which to retrieve the value.

Usage

Use getValueString( ) to retrieve string values from the result set for subsequent processing.

Rule

The specified column must be a String data type.

Return Value

A String value, or null for failure (such as an invalid column number or data type mismatch).

Related Topics

getValueDateString( )

getValueDouble( )

getValueInt( )

moveNext( )

Moves to the next row for the specified child query in the result set.

Syntax

```
public int moveNext(
   String qryName)
```

qryName. Name of the child query that generated the result set.

Usage

Use moveNext( ) when iterating through rows in the result set to retrieve the contents of the next sequential row.

Rule

The specified child query must exist in the result set.

Return Value

An int value for success (zero) or failure (non-zero, such as reaching the end of the result set). If the current row is the last row in the result set, calling moveNext( ) returns a non-zero int.

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IHierQuery Interface (deprecated)

moveTo( )

Moves to the specified row for the specified child query in the result set.

Syntax

```
public moveTo(
   String qryName,
   int nRow)
```

qryName. Name of the child query that generated the result set.

nRow. Number of the row in the result set to move to. The number of the first row in the result set is 1, the second row is 2, and so on.

Usage

Use moveTo( ) to move the internal cursor to a specific row in the result set, skipping over rows to be excluded from processing.

Rules

- The specified child query must exist in the result set.
- The specified row number must exist in the result set.
- If RS_BUFFERING is turned on, AppLogic can move forward and backwards in the result set. However, if RS_BUFFERING is not turned on, AppLogic can move forward to subsequent rows only. AppLogic cannot return to rows that have been processed previously.

Tip

For certain database drivers, this operation may be very slow and should be avoided if possible.

Return Value

An int value for success (zero), or failure (non-zero, such as an invalid row number).

Related Topics

createHierQuery( ) in the AppLogic class (deprecated)

IHierQuery Interface (deprecated)

IListRowSet Interface

IListRowSet is an extension to the javax.sql.RowSet interface that provides the methods necessary to populate a listbox in a JSP. Although anyone developing a iPlanet Application Server application can use IListRowSet, this interface is typically used in components generated by Netscape Application Builder.

IListRowSet replaces the IListDataSet interface from iAB 3.0.

Package

com.netscape.server.servlet.extension

Methods

Method	Description
addListItem( )	Adds an item to a list.
setListSelection( )	Sets the item that appears as the default selection in a list.

Related Topics

com.netscape.server.servlet.extension.IRowSet2 interface,
javax.sql.RowSet interface

addListItem( )

Adds an item to a list.

Syntax

```
public abstract void addListItem(String label, String value)
```

label. The label for the list entry; this is not necessarily the name of the item as it appears in the list.

value. The value assigned to the label.

Usage

Use addListItem( ) with setListSelection( ) when working with an IListRowSet to create a selection list or a group of radio buttons.

Example

The following code fragment creates a data set as a list:

```
IListRowSet ds =
BaseUtils.createListRowSet(request,response,"myList");

ds.addListItem("Star Trek", "1");

ds.addListItem("Babylon 5", "2");

ds.addListItem("Red Dwarf", "3");

ds.addListItem("Crusade", "4");

ds.setListSelection("2"); // Babylon 5 is the default selection
```

To display this list on a web page, you might create a JSP containing the following code:

```
<SELECT NAME="TVShow">

%gx type=tile id=myList%

<OPTION VALUE="%gx type=cell id=myList.value% %/gx%"

%gx

type=cell

id=myList.isEqualToExpression(

   myList.value=myList.ListSelection)%SELECTED

%/gx%

>


%gx type=cell id=myList.label%%/gx%

%/gx%

</SELECT>
```

setListSelection( )

Sets the item that appears as the default selection in a list.

Syntax

```
public abstract void setListSelection(String value)
```

value. The value (not the label) in the list that appears as the default selection.

Usage

Use setListSelection( ) with addListItem( ) when working with data sets to create a selection list or a group of radio buttons.

ILock Interface

The ILock interface provides concurrency control for objects operating in a multithreaded environment (for example, in applications that use distributed state).

AppLogics use locks to protect objects during concurrent operations. For example, state and session nodes implement this interface. Applications that access state or session data concurrently must synchronize using the methods in this interface.

A lock has the following attributes:

- A lock mode. You can specify an exclusive or shared lock. An exclusive lock prevents other threads from accessing a locked object. You can also use the lock mode to specify that an operation may continue even if the desired locking mode is not available.
- A caller ID. This setting provides a unique identifer for the caller that places or removes a lock. The identifier is an array of bytes.

The ILock interface defines methods for locking and unlocking objects. It also defines a method for changing the lock mode.

Package

com.kivasoft

Methods

Method	Description
changeMode( )	Changes the lock mode of a currently locked object. This method is not available for the lock interface implemented by state and session objects.
lock( )	Locks an object.
unlock( )	Unlocks a previously locked object.

changeMode( )

Changes the lock on an object.

NOTE This method is not supported for locks on state and session nodes. State and session support only one lock mode, GXLOCK_EXCL, which cannot be changed.

Syntax

```
public int changeMode(
    int dwOldMode,
    int dwNewMode,
    byte[] pID,
    int nSize);
```

dwOldMode. Current lock mode applied to an object. The mode is one of GXLOCK_EXCL (exclusive lock) or GXLOCK_SHARE (shared lock).

dwNewMode. New locking mode, one of GXLOCK_EXCL (exclusive lock) or GXLOCK_SHARE (shared lock). Optionally, the mode may also include GXLOCK_NOBLOCK if the operation should be allowed to continue if the desired locking mode is not available. If GXLOCK_NOBLOCK is not specified, then a thread is blocked if the desired locking mode is not available.

pID. ID of the caller requesting the change to the lock. This value is read only.

nSize. Size of the identifier.

Usage

Use changeMode( ) to change a lock on an object.

Return Value

An integer indicating success or failure of the lock mode change operation.

lock( )

Locks an object.

Syntax

```
public int lock(
    int dwFlags,
    byte[] pID,
    int nSize);
```

dwFlags. Locking mode, one of GXLOCK_EXCL (exclusive lock) or GXLOCK_SHARE (shared lock). Optionally, the mode may also include GXLOCK_NOBLOCK if the operation should be allowed to continue if the desired locking mode is not available. If GXLOCK_NOBLOCK is not specified, then a thread is blocked if the desired locking mode is not available.

GXLOCK_EXCL is the only mode currently supported for locking a state or session node. You cannot specify GXLOCK_NOBLOCK for state and session nodes.

pID. ID of the caller requesting the lock. This value is a byte array. For state and session objects that implement the locking interface, you can pass in a null value for pID because these implementations automatically use the ID of the calling thread for pID.

nSize. Size of the identifier.

Usage

Use lock( ) to lock an object.

Rules

When you lock certain kinds of nodes, the following rules apply:

- After locking a parent state node, do not create or delete a child node under it.
- After locking a state or session node, do not delete the node.

Return Value

An integer indicating success or failure of the lock operation.

Example

The following code shows how to lock and unlock a state node:

```
IState2 marketnews = cacheroot.getStateChild("mktnews");
if (marketnews == null)
   return;

// we expect marketnews state node to be accessed concurrently
// let's lock it

ILock l = null;
int ret;
if (marketnews instanceof ILock)
{
   l = (ILock)marketnews;
   ret = l.lock(GXLOCK.GXLOCK_EXCL, null, 0);
   if (ret != GXE.SUCCESS)
   {
      log("lock error");
```

```
        return;
    }
}
else
{
    log("IState2 object doesn't implement the ILock interface");
    return;
}
if (l == null)
    return;

// we now have the node locked in exclusive mode
// ..... do work ..........
// and unlock the node

ret = l.unlock(GXLOCK.GXLOCK_EXCL, null, 0);
if (ret != GXE.SUCCESS)
    log("unlock error");
```

Related Topics

AppLogic or ISession2 classes

unlock( )

Unlocks a previously locked object.

Syntax

```
public int unlock(
    int dwFlags,
    byte[] pID,
    int nSize);
```

dwFlags. The locking mode previously used to lock the object, either GXLOCK_EXCL (exclusive lock), or GXLOCK_SHARE (shared lock).

GXLOCK_EXCL is the only mode currently supported for unlocking a state or session node.

pID. The ID of the caller that requests lock removal. This value is a byte array. The ID must match the ID with which you set the lock.

Usually you pass in the ID of the executing thread that requests the lock. For state and session objects that implement the locking interface, you can pass in a null value for pID because these implementations automatically use the ID of the calling thread for pID.

nSize. Size of the identifier.

Usage

Use unlock( ) to remove a lock on an object.

Return Value

An integer indicating success or failure of the unlock operation.

Example

The following code shows how to lock and unlock a state node:

```
IState2 marketnews = cacheroot.getStateChild("mktnews");
if (marketnews == null)
   return;

// we expect marketnews state node to be accessed concurrently
// let's lock it

ILock l = null;
int ret;
if (marketnews instanceof ILock)
{
   l = (ILock)marketnews;
   ret = l.lock(GXLOCK.GXLOCK_EXCL, null, 0);
   if (ret != GXE.SUCCESS)
   {
      log("lock error");
      return;
   }
}
else
{
   log("IState2 doesn't implement the ILock interface");
   return;
}
if (l == null)
   return;

// we now have the node locked in exclusive mode
// ..... do work ..........
// and unlock the node

ret = l.unlock(GXLOCK.GXLOCK_EXCL, null, 0);
if (ret != GXE.SUCCESS)
   log("unlock error");
```

Related Topics

AppLogic or ISession2 classes

IMailbox Interface

The IMailbox interface represents an electronic mailbox used for communicating with incoming and outgoing electronic mail. IMailbox provides methods for opening and closing a mailbox, as well as for receiving and sending mail messages. You must have access to either an SMTP or POP mail server.

To create an instance of the IMailbox interface, use createMailbox( ) in the AppLogic class (deprecated), as shown in the following example:

```
IMailbox mb = createMailbox("SMTPHostName", "myUserName",
     "pass7878", "myEmailAddr@myOrg.com");
```

Package

com.kivasoft

Methods

Method	Description
close( )	Closes an open electronic mailbox session.
open( )	Opens a session with the mail server.
retrieve( )	Retrieves unread electronic mail messages from the inbox.
retrieveCount( )	Counts the number of available unread electronic mail messages in the inbox.
retrieveReset( )	Resets the status of retrieved messages in the mailbox from read to unread and abandons (rolls back) any message deletions.
send( )	Sends an electronic mail message to one or more mail addresses.

Related Topics

createMailbox( ) in the AppLogic class (deprecated)

close( )

Closes an open electronic mailbox session.

Syntax

```
public int close()
```

Usage

Use close( ) to close a mailbox session and commit changes on the mail server, if applicable. If sessions are open on both the POP and SMTP server, close( ) terminates both sessions.

Closing a session does not terminate the IMailbox object. The AppLogic can later reopen a session using open( ).

Rule

The AppLogic can only close a mailbox session that is open.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Define the string parameters that will be passed
// to IMailbox methods
String sendhost = "smtp.kivasoft.com";
String recvhost = "pop.kivasoft.com";
String user = "eugene";
String pswd = "eugenesSecretPassword";
String useraddr = "eugene@kivasoft.com";
String sendTo[] = {"friend@otherhost.net", null};
String mesg = "Hi Friend, How are you?";


public void sendMail()
{
   // Create an IMailbox instance
   IMailbox sendMB;
   sendMB = createMailbox(sendhost, user, pswd, useraddr);


   if (sendMB != null) // sendMB successfully created
   {
      // Open a session with the mail server
      if (sendMB.open(GX_MBOX_OPEN_FLAG.OPEN_SEND)== GXE.SUCCESS)
```

```
      {
         // Send a mail message
         sendMB.send(sendTo,mesg);

         // Close the mailbox session
         sendMB.close();
      }
   }
}
```

Related Topics

open( )

createMailbox( ) in the AppLogic class (deprecated)

open( )

Opens a session with the mail server.

Syntax

```
public int open(
   int dwFlag)
```

dwFlag. Access level used to open the mailbox. Specify one of the following options:

- GX_MBOX_OPEN_FLAG.OPEN_RECV to receive emails. Sets up a session with the POP server only.
- GX_MBOX_OPEN_FLAG.OPEN_SEND to send emails. Sets up a session with the SMTP server only.
- GX_MBOX_OPEN_FLAG.OPEN_SEND |GX_MBOX_OPEN_FLAG.OPEN_RECV to send and receive emails.

Usage

Use open( ) to explicitly open a session with the mail server after instantiating the IMailbox object. Alternatively, the AppLogic can open a session after having closed a previous session using close( ).

Depending on the setting of the dwFlag parameter, open( ) starts a session on the SMTP server only, on the POP server only, or on both servers at once (two separate sessions).

Rule

The AppLogic must call open( ) before calling other methods.

Tip

To conserve system resources, use only the access level you need. For example, if the AppLogic will only be sending electronic mail messages, specify GX_MBOX_OPEN_FLAG.OPEN_SEND, not GX_MBOX_OPEN_FLAG.OPEN_SEND |GX_MBOX_OPEN_FLAG.OPEN_RECV.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Define the string parameters that will be passed
// to IMailbox methods
String sendhost = "smtp.kivasoft.com";
String recvhost = "pop.kivasoft.com";
String user = "eugene";
String pswd = "eugenesSecretPassword";
String useraddr = "eugene@kivasoft.com";
String sendTo[] = {"friend@otherhost.net", null};
String mesg = "Hi Friend, How are you?";


public void sendMail()
{
   // Create an IMailbox instance
   IMailbox sendMB;
   sendMB = createMailbox(sendhost, user, pswd, useraddr);


   if (sendMB != null) // sendMB successfully created
   {
```

```
      // Open a session with the mail server
      if (sendMB.open(GX_MBOX_OPEN_FLAG.OPEN_SEND))
      {
         // Send a mail message
         sendMB.send(sendTo,mesg);

         // Close the mailbox session
         sendMB.close();
      }
   }
}
```

Related Topics

send( )

createMailbox( ) in the AppLogic class (deprecated)

retrieve( )

Retrieves electronic mail messages from the inbox.

Syntax

```
public IValList retrieve(
   boolean bLatest,
   boolean bDelete)
```

bLatest. Specify true to retrieve the latest unread messages. Specify false to retrieve all messages in the inbox.

bDelete. Specify true to delete retrieved messages when the mailbox session is closed. Specify false to leave the retrieved messages on the mail server.

Usage

Use retrieve( ) to get unread messages from the inbox. Once retrieved, messages are marked as READ.

Rule

To use retrieve( ), the AppLogic must have first opened the mailbox session using open( ) and have specified either GX_MBOX_OPEN_FLAG.OPEN_RECV or GX_MBOX_OPEN_FLAG.OPEN_SEND | GX_MBOX_OPEN_FLAG.OPEN_RECV as the dwFlag parameter.

Tip

AppLogic can use retrieveReset( ) to undo changes (deletes, read flags) to messages in the inbox.

Return Value

IValList that contains the retrieved messages, or null for failure.

Example

```
// Create mailbox object to connect to a POP mail server
IMailbox recvMB;


public void recvMail()
{
   // Only check messages received after the last open
   BOOL Latest = true;


   // Remove retrieved messages from the mail server
   BOOL Delete = true;


   // Create an IMailbox instance
   IMailbox recvMB;
   recvMB = createMailbox(recvhost, user, pswd, useraddr);
   if (recvMB != null)
   {
      if (recvMB.open(GX_MBOX_OPEN_FLAG.OPEN_RECV))
      {
         // Count the number of new messages
         int numMsgs = recvMB.retrieveCount();
         if(numMsgs > 0)
```

```
         {
            IValList mesgList;
            // Retrieve the new messages
            mesgList = recvMB.retrieve(Latest,Delete);
            // Use IValList methods to iterate through
            // the returned IValList. The keys in the
            // IValList are the message numbers. The
            // values are the email messages as strings
         }
         recvMB.close();
      }
   }
}
```

Related Topics

open( )

createMailbox( ) in the AppLogic class (deprecated)

retrieveCount( )

Counts the number of unread electronic mail messages in the inbox.

Syntax

```
public int retrieveCount()
```

Usage

Before calling retrieve( ), use retrieveCount( ) to count the number of retrievable messages in the inbox. The AppLogic might do this to avoid retrieving an empty inbox. If the AppLogic iterates through the messages after they have been retrieved, the AppLogic can call retrieveCount( ) to determine the maximum number of iterations required to process all available inbox messages.

Rule

To use retrieveCount( ), the AppLogic must have first opened the mailbox session using open( ) and have specified either GX_MBOX_OPEN_FLAG.OPEN_RECV or GX_MBOX_OPEN_FLAG.OPEN_SEND | GX_MBOX_OPEN_FLAG.OPEN_RECV as the dwFlag parameter.

Return Value

The number of available unread electronic mail messages in the inbox. The retrieveCount( ) method returns 0 for no messages and a negative number if an error ocurred.

Example

```
// Create mailbox object to connect to a POP mail server
IMailbox recvMB;


public void recvMail()
{
   // Only check messages received after the last open
   BOOL Latest = true;



   // Remove retrieved messages from the mail server
   BOOL Delete = true;
   // Create an IMailbox instance
   IMailbox recvMB;
   recvMB = createMailbox(recvhost, user, pswd, useraddr);

   if (recvMB != null)
   {
      if (recvMB.open(GX_MBOX_OPEN_FLAG.OPEN_RECV))
      {
         // Count the number of new messages
         int numMsgs = recvMB.retrieveCount();
         if(numMsgs > 0)
         {
            IValList mesgList;
            // Retrieve the new messages
            mesgList = recvMB.retrieve(Latest,Delete);
```

```
            // Use IValList methods to iterate through
            // the returned IValList. The keys in the
            // IValList are the message numbers. The
            // values are the email messages as strings
         }
         recvMB.close();
      }
   }
}
```

Related Topics

open( )

retrieve( )

createMailbox( ) in the AppLogic class (deprecated)

retrieveReset( )

Resets the status of retrieved messages in the mailbox from read to unread and abandons (rolls back) any message deletions.

Syntax

```
public int retrieveReset()
```

Usage

Use retrieveReset( ) to undo any changes made as a result of retrieving inbox messages with retrieve( ).

Rules

- To use retrieveReset( ), the AppLogic must have first opened the mailbox session using open( ) and have specified either GX_MBOX_OPEN_FLAG.OPEN_RECV or GX_MBOX_OPEN_FLAG.OPEN_SEND | GX_MBOX_OPEN_FLAG.OPEN_RECV as the dwFlag parameter.
- Before calling retrieveReset( ), the AppLogic must first call retrieve( ).
- To abandon changes made with retrieve( ), AppLogic must call retrieveReset( ) before calling close( ) or terminating the session.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
open( )

retrieve( )

createMailbox( ) in the AppLogic class (deprecated)

send( )

Sends an electronic mail message to one or more mail addresses.

Syntax
```
public int send(
   String[] ppTo,
   String pMesg)
```

ppTo. A list of email addresses, to which you want to send e-mail. The address or addresses must be supplied in a null-terminated array.

pMesg. Text of the electronic mail message. Use Internet mail formatting conventions for specifying advanced features in the message text, such as CC: or BCC: addresses, the Subject header, uuencode, MIME attachments, receipt notification, and so on. For syntax specifications, see your POP and SMTP protocol documentation.

Rules
- To use send( ), the AppLogic must have first opened the mailbox session using open( ) and have specified either GX_MBOX_OPEN_FLAG.OPEN_SEND or GX_MBOX_OPEN_FLAG.OPEN_SEND | GX_MBOX_OPEN_FLAG.OPEN_RECV as the dwFlag parameter.
- The specified addresses must be valid Internet mail addresses.
- The specified message text must follow POP and SMTP protocol conventions.

Tip
The send( ) method automatically includes the FROM: address that the AppLogic specified in the pUserAddr parameter of createMailbox( ) in the AppLogic class (deprecated).

Return Value
GXE.SUCCESS if the method succeeds.

Example

```
// Define the string parameters that will be passed
// to IMailbox methods
String sendhost = "smtp.kivasoft.com";
String recvhost = "pop.kivasoft.com";
String user = "eugene";
String pswd = "eugenesSecretPassword";
String useraddr = "eugene@kivasoft.com";
String sendTo[] = {"friend@otherhost.net", null};
String mesg = "Hi Friend, How are you?";

public void sendMail()
{
   // Create an IMailbox instance
   IMailbox sendMB;
   sendMB = createMailbox(sendhost, user, pswd, useraddr);

   if (sendMB != null) // sendMB successfully created
   {
      // Open a session with the mail server
      if (sendMB.open(GX_MBOX_OPEN_FLAG.OPEN_SEND))
      {
         // Send a mail message
         sendMB.send(sendTo,mesg);
         // Close the mailbox session
         sendMB.close();
      }
   }
}
```

Related Topics

open( ),
retrieve( )

createMailbox( ) in the AppLogic class (deprecated)

IObject Interface *(deprecated)*

The IObject interface is not necessary in the new application model. This interface is deprecated and is provided for backward compatiblity only.

The IObject interface is the base interface for all Netscape Application Server Java interfaces. Generally, iPlanet Application Server applications do not use this interface directly; they use the specialized derived interfaces instead.

Package

com.kivasoft

IOrder Interface (*deprecated*)

IOrder is deprecated and is provided for backward compatibility only. New applications developed according to the servlet-JSP programming model do not need the functionality provided by IOrder.

The IOrder interface represents the current processing status of an asynchronous operation. IOrder provides methods for obtaining the status and return code of an asynchronous operation.

To run an asynchronous database operation, the AppLogic must specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC as the flags parameter in any of the following methods:

- executeQuery( ) in the IDataConn Interface (deprecated)
- addRow( ), deleteRow( ), or updateRow( ) in the ITable Interface (deprecated)

To create an instance of the IOrder interface for an asynchronous query, use getOrder( ) in the IResultSet Interface (deprecated), as shown in the following example:.

```
IResultSet rs;

IOrder ord

ord=rs.getOrder();
```

Package

com.kivasoft

Methods

getState( )	Returns the processing status of the asynchronous operation on the database server: active, done, canceled, or unknown.

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

getOrder( ) in the IResultSet Interface (deprecated)

GX.WaitForOrder( )

getState( )

Returns the processing status of the asynchronous operation.

Syntax

```
public getStateIOrder getState()
```

Usage

Use getState( ) to return status information to use in error-handling code.

Return Value

A getStateIOrder object with three variables, each containing status and result code values. The three variables are described in the following table:

Variable	Description
pdwState	Contains one of the following status codes: • GXORDER_STATE_ACTIVE • GXORDER_STATE_CANCEL • GXORDER_STATE_DONE • GXORDER_STATE_UNKNOWN

Variable	Description
pdwResult	Contains the result code of an asynchronous database operation when the operation is done or canceled.
pGuess	Contains the estimate about the current completion percentage of the operation.

Example

```
Orders[] = new IOrder[1];
int nOrder;

Orders[0] = newRequestAsync(asyncGUIDStr, valIn, valOut);
if (Orders[0] != null)
{
   log("Successfully invoked async AppLogic\n");

   // wait for async applogic to finish (max 100 seconds)
   nOrder = GX.WaitForOrder(Orders, context, 100);
   if (nOrder >= 0)
   {
       return result("Error in executing async request:
            order wait returned an error");
   }
   else
   {
      getStateIOrder state = Orders[0].getState();
      if (state == null || state.pdwResult != GXE.SUCCESS)
         return result("Error in executing async
              request");
   }
}
else
```

```
{
   log("Failed to invoke async AppLogic\n");
}
```

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

getOrder( ) in the IResultSet Interface (deprecated)

GX.WaitForOrder( )

IPreparedQuery Interface *(deprecated)*

IPreparedQuery is deprecated and is provided for backward compatibility only. New applications should use the java.sql.PreparedStatement interface from the JDBC Core API.

The IPreparedQuery interface represents a prepared flat query. An IPreparedQuery object contains a SQL statement that has been compiled. This is what makes a statement "prepared." An AppLogic uses a prepared query when it needs to execute a SQL statement multiple time with different parameters.

For example, if an AppLogic runs an INSERT statement several times, each time with a different set of values to insert into the table, using a prepared query involves the following steps:

1. Prepare (compile) the INSERT statement with placeholder parameters whose values will be specified later.
2. Specify a set of parameter values.
3. Execute the prepared query.
4. Specify another set of parameter values.
5. Execute the prepared query.

By preparing the SQL statement, the database needs to compile the statement only once. Without prepared statements, the database must recompile each statement every time it is executed, which is less efficient.

To create an instance of the IPreparedQuery interface, use prepareQuery( ) in the IDataConn Interface (deprecated), as shown in the following example:

```
IPreparedQuery pqry;
pqry = conn.prepareQuery(0, qry, null, null);
```

Package

com.kivasoft

Methods

Name	Description
execute( )	Executes a prepared query.
setParams( )	Specifies the parameters and flags for a prepared query.

Example

```
// Create the data connection

IDataConn conn;

conn = createDataConn(0,GX_DA_DAD_DRIVERS.GX_DA_DRIVER_DEFAULT,
"Orders", "Orders", "user", "password");

// Create the flat query and prepared query objects

IQuery qry = createQuery();

IPreparedQuery pqry;

IResultSet rs1, rs2;

// Set up the INSERT statement

qry.setSQL("INSERT INTO TABLE Products (ProductName,
QuantityPerUnit) VALUES (:name, :quant)");

// Prepare the flat query

pqry = conn.prepareQuery(0, qry, null, null);

// Specify a set of query parameters, then execute

IValList params = GX.CreateValList()

params.setValString(":name","Chicken Dumplings");

params.setValString(":quant", "48 packages");

rs1 = pqry.execute(0, params, null, null);

. . . process rs1 . . .

// Specify different set of query parameters, then execute

params.setValString(":name", "Rice Noodles");
```

```
params.setValString(":quant", "96 packages");
rs2 = pqry.execute(0, params, null, null);
. . . process rs2 . . .
```

Related Topics

prepareQuery( ) in the IDataConn Interface (deprecated)

execute( )

Executes a prepared query.

Syntax

```
public IResultSet execute(
   int dwFlags,
   IValList pParams,
   ITrans pTrans,
   IValList pProps)
```

dwFlags. Specifies flags used to execute this prepared query. To activate result set buffering, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_RS_BUFFERING. Otherwise, specify zero.

pParams. IValList object that contains parameters to pass to the prepared query. Parameters are used to execute the query.

pTrans. ITrans object that contains the transaction associated with this query, or null for no transaction.

pProps. *I*ValList object that contains query properties, or null for no properties. After instantiating an object of the IValList interface, set any of the following properties:

- RS_BUFFERING turns on result set buffering when set to “TRUE”.
- RS_INIT_ROWS specifies the initial size of the buffer, in number of rows. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED and result set buffering will be turned off.
- RS_MAX_ROWS specifies the maximum number of rows for the buffer. If the result set size exceeds this setting, a fetchNext( ) call will return the error GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED and result set buffering will be turned off.
- RS_MAX_SIZE specifies the maximum number of bytes for the buffer.

If RS_BUFFERING is enabled and if the optional parameters are not specified, the global values in the registry are used instead.

Usage

Use execute( ) to run a prepared query. If the command contains parameters, instantiate an IValList object and use setValString( ) or setValInt( ) in the IValList interface to specify the parameter values to pass to the command.

Return Value

IResultSet object, or null for failure (such as an invalid parameter).

Example 1

```
// Create the data connection

IDataConn conn;

conn = createDataConn(0,GX_DA_DAD_DRIVERS.GX_DA_DRIVER_DEFAULT,
"Orders", "Orders", "user", "password");

// Create the flat query and prepared query objects

IQuery qry = createQuery();

IPreparedQuery pqry;

IResultSet rs;

// Set up the INSERT statement

qry.setSQL("INSERT INTO TABLE Products (ProductName,
QuantityPerUnit) VALUES (:name, :quant)");

// Prepare the flat query

pqry = conn.prepareQuery(0, qry, null, null);

// Specify a set of query parameters, then execute

IValList params = GX.CreateValList();

params.setValString(":name","Chicken Dumplings");

params.setValString(":quant", "48 packages");

rs = pqry.execute(0, params, null, null);

. . . process result set . . .

// Specify different set of query parameters, then execute

params.setValString(":name", "Rice Noodles");

params.setValString(":quant", "96 packages");

rs = pqry.execute(0, params, null, null);
```

```
. . . process result set . . .
```

Example 2

```
// Set up result set buffering for prepared query
// . . . data connection and flat query already set up . . .
IPreparedQuery pqry;
IValList props;
props = GX.CreateValList();
// Turn on result set buffering
props.setValString("RS_BUFFERING", "TRUE");
// Specify the maximum number of rows to buffer
props.setValInt("RS_MAX_ROWS", 50);
pqry = conn.prepareQuery(0, qry, null, props);
params.setValString(":name","Chicken Dumplings");
params.setValString(":quant", "48 packages");
IResultSet rs
rs = pqry.execute(GX_DA_EXECUTEQUERY_FLAGS.GX_DA_RS_BUFFERING,
params, null, null);
. . . process result set . . .
```

Related Topics

IValList Interface (deprecated)

ITrans Interface (deprecated)

prepareQuery( ) in the IDataConn Interface (deprecated)

setParams( )

Specifies the parameters for a prepared query.

Syntax

```
public int setParams(
   int dwFlags
   IValList pParams)
```

dwFlags. Specify zero (0).

pParams. Pointer to an IValList object that contains parameters to pass to the prepared query.

Usage

To pass parameters to the prepared query using setParams( ), you must pass NULL for the pParams parameter in execute( ).

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

IValList Interface (deprecated)

ITrans Interface (deprecated)

prepareQuery( ) in the IDataConn Interface (deprecated)

IQuery Interface *(deprecated)*

IQuery is deprecated and is provided for backward compatibility only. New applications should use the java.sql.Statement interface from the JDBC Core API. Note that IQuery is independent of a database connection, whereas JDBC's Statement interface is derived from a connection.

The IQuery interface represents a flat query. IQuery provides methods for specifying and obtaining the criteria used to select data from a data source. The AppLogic uses IQuery member methods to specify all parts of the SQL SELECT statement, including the SELECT, FROM, GROUP BY, HAVING, ORDER BY, and WHERE clauses.

To run a flat query, the AppLogic performs the following steps:

1. Creates an IQuery object using createQuery( ) in the AppLogic class (deprecated).
2. Specifies query criteria using methods in the IQueryinterface.
3. Executes the query, passing the loaded IQuery object to executeQuery( ) in the IDataConn Interface (deprecated).
4. Processes the result set using methods in the IResultSet Interface (deprecated).

The AppLogic can also use IQuery methods to obtain information about query criteria when the criteria are unknown. Before executing the query on the data source, the AppLogic can evaluate and, if necessary, dynamically change the query criteria.

To create an instance of the IQuery interface, use the createQuery( ) method in the AppLogic class, as shown in the following example:.

```
IQuery qry;
qry = createQuery();
```

Package

com.kivasoft

Methods

Method	Description
getFields( )	Returns a comma-separated list of arbitrary SQL expressions or columns to be included in the result set of the query.
getGroupBy( )	Returns the GROUP BY clause of the query.
getHaving( )	Returns the HAVING clause of the query.
getOrderBy( )	Returns the ORDER BY clause of the query.
getSQL( )	Returns the SQL pass-through statement associated with the query.
getTables( )	Returns a comma-separated list of tables in the FROM clause of the query.
getWhere( )	Returns the WHERE clause of the query.
setFields( )	Specifies the list of columns and computed fields to be included in the result set of the query. Required method when writing a query.
setGroupBy( )	Specifies the GROUP BY clause of the query, determining how rows are grouped and calculated.
setHaving( )	Specifies the HAVING clause of the query, determining which aggregate rows qualify for inclusion in the result set.
setOrderBy( )	Specifies the ORDER BY clause of the query, determining how rows are sorted in the result set.
setSQL( )	Specifies the SQL statement to be passed directly to the data source.
setTables( )	Specifies the FROM clause of the query, identifying one or more tables to be queried. Required method when writing a query.

Method	Description
setWhere( )	Specifies the WHERE clause of the query, determining which rows qualify for inclusion in the result set.

Example 1

```
// Create the flat query object
IQuery qry=createQuery();

// Set up the query
qry.setTables("CTLcust");
qry.setFields("CustomerID, Customer");
qry.setWhere("Customer"+"="+String.valueOf(custId));

// Execute the query
IResultSet rs=conn.executeQuery(0, qry, null, null);

// Check for a non empty result
if((rs!=null)&&(rs.getRowNumber()>0))
   return result("Sorry, this user ("+
   firstName+" "+lastName+") already exists");
. . . process the result set . . .
```

Example 2

```
// Create a database connection
IDataConn conn;

if((conn = openBugDB())==null) return 0;
// Pull all the parms we need from the valIn

if((wantedUser = valIn.getValString("WantedUser"))==null)
    return result("Missing information from form.");
```

```
// Create the flat query object
IQuery qry = createQuery();


// Set up query conditions
qry.setTables("CTLusers");
qry.setFields("Name, Password, UserType, AccessLevel, Email,
ExtID");
query.setWhere("UserID"+"="+wantedUser.hashCode());
. . . execute the query . . .
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

executeQuery( ) in the IDataConn Interface (deprecated)

IResultSet Interface (deprecated)

getFields( )

Returns a comma-separated list of arbitrary SQL expressions or columns to be included in the result set of the query.

Syntax

```
public IBuffer getFields()
```

Usage

In a SQL SELECT statement, the first clause specifies the SELECT keyword as well as the list of columns to be retrieved in the result set.

Use getFields( ) when the requested columns in a query are unknown, such as when using a query from another source. The AppLogic can analyze this list to determine the names of the columns as well as the order in which they will appear in the result set. Before executing or re-executing the query, the AppLogic can evaluate and, if necessary, dynamically change columns and column order in the query by calling setFields( ).

Tips

- To use a query obtained from another source such as a file, the AppLogic can call getFields( ) and other getXXXX( ) member methods to test the query statement before submitting it to the server for processing. The AppLogic can then use the setXXXX( ) member methods to change the statement and avoid lengthy queries or syntax errors.
- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains a comma-separated list of columns that the query defines for the result set, starting with the first column and proceeding sequentially, left to right.

Example

```
// Get the columns specified in the query's SELECT clause

String qryFields;

qryFields = Util.toString(qry.getFields());
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

getGroupBy( )

Returns the GROUP BY clause of the query.

Syntax

```
public IBuffer getGroupBy()
```

Usage

In a SQL SELECT statement, the GROUP BY clause specifies rows to summarize into aggregate rows using column functions (such as SUM or MAX) or column names.

Use getGroupBy( ) when the GROUP BY clause of the query is unknown, such as when using a query from another source. Before executing the query, the AppLogic can evaluate and, if necessary, dynamically change the GROUP BY clause by calling setGroupBy( ).

Tips

- To use a query obtained from another source such as a file, the AppLogic can call getGroupBy( ) and other getXXXX( ) member methods to test the query statement before submitting it to the server for processing. The AppLogic can then use the setXXXX( ) member methods to change the statement and avoid lengthy queries or syntax errors.
- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains the GROUP BY clause of the query, or null for failure.

Example

```
// Obtain the GROUP BY clause specified in the query

String qryGroup;

qryGroup = Util.toString(qry.getGroupBy());
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

getHaving( )

Returns the HAVING clause of the query.

Syntax

```
public IBuffer getHaving()
```

Usage

In a SQL SELECT statement, the HAVING clause specifies which of the aggregate rows returned by the GROUP BY clause are selected for the result set.

Use getHaving( ) when the HAVING clause of the query is unknown, such as when using a query from another source. Before executing the query, the AppLogic can evaluate and, if necessary, dynamically change the HAVING clause by calling setHaving( ).

Tips

- To use a query obtained from another source such as a file, the AppLogic can call getHaving( ) and other getXXXX( ) member methods to test the query statement before submitting it to the server for processing. The AppLogic can then use the setXXXX( ) member methods to change the statement and avoid lengthy queries or syntax errors.
- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains the HAVING clause of the query, or null for failure.

Example

```
// Obtain the HAVING clause specified in the query

String qryHaving;

qryHaving = Util.toString(qry.getHaving());
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

getOrderBy( )

Returns the ORDER BY clause of the query.

Syntax

```
public IBuffer getOrderBy()
```

Usage

In a SQL SELECT statement, the ORDER BY clause specifies one or more columns by which rows in the result set are sorted, as well as whether they appear in ascending or descending ASCII order.

Use getOrderBy( ) when the ORDER BY clause of the query is unknown, such as when using a query from another source. Before executing the query, the AppLogic can evaluate and, if necessary, dynamically change the ORDER BY clause by calling setOrderBy( ).

Rule

Some database vendors have restrictions on the ordering and usage of ORDER BY clauses. Read your database vendor's documentation carefully and test queries to ensure that they return the desired results.

Tips

- To use a query obtained from another source such as a file, the AppLogic can call getOrderBy( ) and other getXXXX( ) member methods to test the query statement before submitting it to the server for processing. The AppLogic can then use the setXXXX( ) member methods to change the statement and avoid lengthy queries or syntax errors.
- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains the ORDER BY clause of the query, or null for failure.

Example

```
// Obtain the ORDER BY clause specified in the query

String qryOrder;

qryOrder = Util.toString(qry.getOrderBy());
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

getSQL( )

Returns the SQL pass-through statement associated with the query.

Syntax

```
public IBuffer getSQL()
```

Usage

Use getSQL( ) when the query string is unknown, such as when using a query from another source. Before executing the query, the AppLogic can dynamically change the SQL statement by calling setSQL( ).

Rule

If a query is set using setSQL( ) as well as the setXXXX( ) methods, the setSQL( ) string will be executed, not the string specified by setXXXX( ).

Tips

- To use a query obtained from another source such as a file, the AppLogic can call getSQL( ) and other getXXXX( ) member methods to test the query statement before submitting it to the server for processing. The AppLogic can then use the setXXXX( ) member methods to change the statement and avoid lengthy queries or syntax errors.
- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains the SQL pass-through statement of the query (as a single concatenated string), or null for failure.

Example

```
// Obtain the SELECT statement specified in the query

String qrySelect;

qrySelect = Util.toString(qry.getSQL());
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

getTables( )

Returns a comma-separated list of tables in the FROM clause of the query.

Syntax

```
public IBuffer getTables()
```

Usage

In a SQL SELECT statement, the FROM clause specifies one or more source tables, views, or table aliases to search in the query. In iPlanet Application Builder, the AppLogic can obtain table names only.

Use getTables( ) when the FROM clause of the query is unknown, such as when using a query from another source. Before executing the query, the AppLogic can evaluate and, if necessary, dynamically change the FROM clause by calling setTables( ).

Tips

- To use a query obtained from another source such as a file, the AppLogic can call getTables( ) and other getXXXX( ) member methods to test the query statement before submitting it to the server for processing. The AppLogic can then use the setXXXX( ) member methods to change the statement and avoid lengthy queries or syntax errors.
- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains the FROM clause of the query, or null for failure.

Example

```
// Obtain the FROM clause specified in the query
String qryFrom;
qryFrom = Util.toString(qry.getTables());
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

getWhere( )

Returns the WHERE clause of the query.

Syntax

```
public IBuffer getWhere()
```

Usage

In a SQL SELECT statement, the WHERE clause specifies the search condition and determines which rows in the table are selected for the result set.

Use getWhere( ) when the WHERE clause of the query is unknown, such as when using a query from another source. Before executing the query, the AppLogic can evaluate and, if necessary, dynamically change the WHERE clause by calling setWhere( ).

Tips

- To use a query obtained from another source such as a file, the AppLogic can call getWhere( ) and other getXXXX( ) member methods to test the query statement before submitting it to the server for processing. The AppLogic can then use the setXXXX( ) member methods to change the statement and avoid lengthy queries or syntax errors.

- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains the WHERE clause of the query, or null for failure.

Example

```
// Obtains the WHERE clause specified in the query

String qryWhere;

qryWhere = Util.toString(qry.getWhere());
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

setFields( )

Specifies the list of columns and computed fields to be included in the result set of the query. Required method when writing a query.

Syntax

```
public int setFields(
   String szFields)
```

szFields. List of field names, separated by commas, or an asterisk (*) to include all fields. Extra whitespace characters are ignored. Use the AS keyword to specify field aliases. Defaults to all fields (*).

Usage

In a SQL SELECT statement, the first clause specifies the SELECT keyword as well as the list of columns and computed fields to be retrieved in the result set. The AppLogic can specify field aliases using the AS keyword in the setFields( ) parameter list.

A computed field is the result of an expression using either of the following kinds of expressions:

- **Mathematical functions**, including SQL string, numeric, time, date, system, and data type conversion functions and mathematical operators
- **Aggregate functions**, including SUM, COUNT, MIN, MAX, AVG, to summarize values per column across a group of rows. These functions are commonly used in conjunction with the GROUP BY clause, which the AppLogic can specify using setGroupBy( ).

Rules

- Use ANSI 92 SQL-compliant syntax for the field list.
- Use implementation-specific SQL syntax extensions only on data sources that support them. Using extensions may compromise portability across platforms.
- Any specified column names must appear in one of the tables specified in setTables( ). Table qualified names are permitted, such as `"prod.name,emp.name"`.

Tip

For computed fields, use the AS keyword so that the AppLogic can process the column in the result set by alias name.

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Create the flat query object

IQuery qry = createQuery();

// Set up query conditions

qry.setTables("CTLusers");

// Specify list of fields to retrieve in the result set

qry.setFields("Name, Password, UserType, AccessLevel, Email, ExtID
as ID");

query.setWhere("UserID"+"="+wantedUser.hashCode());

query.setOrderBy("Name");

. . . execute the query . . .
```

Example 2

```
// Specify result set fields, specifying alias for calculation

qry.setFields("ProdID, ProdPrice * ProdQty AS ExtPrice");
```

Example 3

```
// Parentheses specify mathematical operator precedence

qry1.setFields("(product.price * invoice.quantity) / 100 as total,
invoice.id");

qry2.setFields("invoice.id, product.price * 0.06 as tax");
```

Example 4

```
// COUNT and SUM aggregate functions and alias name for result
qry1.setFields("COUNT(invoice.id) as totalSold");
qry2.setFields("SUM(city.population) as urbanPopulation");
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

setGroupBy( )

Specifies the GROUP BY clause of the query, determining how rows are grouped and calculated.

Syntax

```
public int setGroupBy(
   String szGroupBy)
```

szGroupBy. GROUP BY clause of the query, using standard SQL syntax.

Usage

In a SQL SELECT statement, the GROUP BY clause specifies rows to combine using column functions (such as SUM or MAX) or column names. Such groupings are called aggregate rows, which are single rows in a result set that combine data from a group of database rows with one or more column values in common.

Rules

- Use ANSI 92 SQL-compliant syntax for the GROUP BY clause.
- Use implementation-specific SQL syntax extensions only on data sources that support them. Using extensions may compromise portability across platforms.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
IQuery query=createQuery();
qry.setTables("CTLinvoice,CTLcust,CTLinvprod,CTLcatalog");
qry.setFields("FirstName, LastName, BillAddr1, BillAddr2,
      BillCity, BillState, BillZip,
      count(*) as Visits",
      sum(Quantity) as ProductsBought,
```

```
      sum(quantity*price) as AmountSpent,
      quantity/count(*) as ProdsPerVisit,
      sum(quantity*price/count(*) as spentPerVisit")
qry.setWhere("Customer=CustomerId and Invoice=InvoiceId and
      Product=ProductId and
      datePlaced >= start and datePlaced"<=end");
// Group customers by first name
qry.setGroupBy("FirstName);
qry.setOrderBy("Visits desc");
. . . run query . . .
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

setHaving( )

Specifies the HAVING clause of the query, determining which aggregate rows qualify for inclusion in the result set.

Syntax

```
public int setHaving(
   String szGroupBy)
```

szGroupBy. HAVING clause of the query, using standard SQL syntax.

Usage

The HAVING clause is used in conjunction with the aggregate functions (SUM, AVG, and so on) and the GROUP BY clause. In a SQL SELECT statement, the HAVING clause specifies a condition that determines which aggregate rows are selected for the result set. The HAVING clause restricts the number of aggregate rows retrieved in the result set. If unspecified, all aggregate rows will be retrieved.

Rules

- Use ANSI 92 SQL-compliant syntax for the HAVING clause.
- Use implementation-specific SQL syntax extensions only on data sources that support them. Using extensions may compromise portability across platforms.

Tips

- The order in which you specify a HAVING clause, in relation to other query clauses, may affect which records are retrieved in the result set. See your RDBMS server documentation for more information.
- To improve the AppLogic performance, be sure to specify a HAVING or WHERE clause to avoid retrieving rows unnecessarily, especially for large tables.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Set condition on aggregate rows to an

// average amount > 1000

qry.setHaving("AverageAmt > 1000");
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

setOrderBy( )

Specifies the ORDER BY clause of the query, determining how rows are sorted in the result set.

Syntax

```
public int setOrderBy(
   String szOrderBy)
```

szOrderBy. ORDER BY clause of the query, using standard SQL syntax. Supports the ASC and DESC keywords for sorting.

Usage

In a SQL SELECT statement, the ORDER BY clause specifies one or more columns by which rows in the result set are sorted. The AppLogic can also specify whether records appear in ascending (the default) or descending ASCII order using the ASC and DESC keywords, respectively.

Rules

- Use ANSI 92 SQL-compliant syntax for the ORDER BY clause.
- Use implementation-specific SQL syntax extensions only on data sources that support them. Using extensions may compromise portability across platforms.

- Any specified column names must appear in one of the columns specified in setFields( ).
- Some database vendors have restrictions on the ordering and usage of ORDER BY clauses. Read your database vendor's documentation carefully and test queries to ensure that they return the desired results.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
IQuery qry=createQuery();
qry.setTables("CTLinvoice,CTLcust,CTLinvprod,CTLcatalog");
qry.setFields("FirstName, LastName, BillAddr1, BillAddr2,
          BillCity, BillState, BillZip,
          count(*) as Visits",
          sum(Quantity) as ProductsBought,
          sum(quantity*price) as AmountSpent,
          quantity/count(*) as ProdsPerVisit,
          sum(quantity*price/count(*) as spentPerVisit")
qry.setWhere("Customer=CustomerId and Invoice=InvoiceId and
          Product=ProductId and
          datePlaced >= start and datePlaced <= end");
qry.setGroupBy("FirstName");
// Sorts rows in result set by number of visits
// in descending order
qry.setOrderBy("Visits desc");
. . . run query . . .
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

setSQL( )

Specifies the SQL statement to be passed directly to the data source.

Syntax

```
public int setSQL(
    String szSQL)
```

szSQL. SQL statement, using standard SQL syntax, to execute on the target data source. Specify a single, concatenated string. Do not use semicolon (;) characters or other vendor-specific statement delimiters.

Usage

The AppLogic can use setSQL( ) as an alternative to using other iPlanet Application Builder methods, such as constructing queries, inserting, updating, and deleting rows, and managing transactions. The AppLogic can also use setSQL( ) to run specialized SQL statements, such Data Definition Language (DDL) commands, Data Control Language (DCL) commands, and so on.

Rules

- Use ANSI 92 SQL-compliant syntax for the SQL statement.
- Use implementation-specific SQL syntax extensions only on data sources that support them. Using extensions may compromise portability across platforms.
- The AppLogic must be logged in with sufficient privileges to permit any operations requested in the passed-through SQL statement.
- If inserting or updating rows in a table, the AppLogic must specify values that are valid. For example, the AppLogic cannot omit specifying a value for any column defined as NOT NULL and without a DEFAULT value, such as keys.
- Using setSQL( ) overrides all previous calls to setXXXX( ) member methods for this query object. If a query is set using setSQL( ) as well as the setXXXX( ) methods, the setSQL( ) string will be executed, not the string specified by setXXXX( ).
- If the statement is part of a transaction, the AppLogic must first create an instance of the ITrans Interface (deprecated) using createTrans( ) in the AppLogic class (deprecated). The AppLogic must then call begin( ) before executing the statement and, after executing the statement, call commit( ) or rollback( ) as appropriate.

Tip

To determine whether a column is defined as NOT NULL, use getNullsAllowed( ) in the IColumn Interface (deprecated).

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Pass SELECT statement to setSQL()

IQuery qry = createQuery();


qry.setSQL("SELECT empName, empSalary FROM employees WHERE empSalary
> 10000");


IResultSet rs=conn.executeQuery(0, query, null, null);

if(rs!=null) rs.flush();

   System.out.println("Writing to database");
```

Example 2

```
// Pass INSERT command to setSQL()

qry.setSQL("INSERT into employees (empSalary) values (29)");

Example 3

// Pass UPDATE command to setSQL()

qry.setSQL("UPDATE employees SET empSalary = 12000 WHERE empSalary =
10000");
```

Example 3

```
// Pass DELETE command to setSQL()

qry.setSQL("DELETE FROM employees WHERE empSalary < 10000");
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

Vendor documentation regarding SQL programming for the specific data source that is the target of the SQL statement.

setTables( )

Specifies the FROM clause of the query, identifying one or more tables to be queried. Required method when writing a query.

Syntax

```
public int setTables(
   String szTables)
```

szTables. List of table names separated by commas. Whitespace characters are ignored.

Usage

In a SQL SELECT statement, the FROM clause specifies one or more source tables, views, or table aliases to search in the query. In iPlanet Application Builder, the AppLogic can specify table names only.

Rules

- Use ANSI 92 SQL-compliant syntax for the FROM clause.
- Use implementation-specific SQL syntax extensions only on data sources that support them. Using extensions may compromise portability across platforms.
- The AppLogic can specify table names but not table aliases or view names.
- The AppLogic can use the same table several times in a query. To do so, specify a different alias name each time the table is used.

Example 1

```
// Specify invoice, customer, invoice, and catalog tables
IQuery qry=createQuery();
qry.setTables("CTLinvoice,CTLcust,CTLinvprod,CTLcatalog");
qry.setFields("FirstName, LastName, BillAddr1, BillAddr2,
          BillCity, BillState, BillZip,
          count(*) as Visits",
          sum(Quantity) as ProductsBought,
          sum(quantity*price) as AmountSpent,
          quantity/count(*) as ProdsPerVisit,
          sum(quantity*price/count(*) as spentPerVisit")
qry.setWhere("Customer=CustomerId and Invoice=InvoiceId and
          Product=ProductId and
          datePlaced >= start and datePlaced <= end");
qry.setGroupBy("FirstName");
```

Example 2

```
// Specify same table, using as keyword, for different users
qry.setTables("customer, customer as cust2")
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

setWhere( )

Specifies the WHERE clause of the query, determining which rows qualify for inclusion in the result set.

Syntax

```
public int setWhere(
   String szWhere)
```

szWhere. WHERE clause of the query, using standard SQL syntax.

Usage

In a SQL SELECT statement, the WHERE clause specifies the search condition and determines which rows in the table are selected for the result set. The WHERE clause restricts the number of rows retrieved in the result set. If unspecified, all rows in the source table will be retrieved.

Rules

- Use ANSI 92 SQL-compliant syntax for the WHERE clause.
- Use implementation-specific SQL syntax extensions only on data sources that support them. Using extensions may compromise portability across platforms.

Tip

To improve AppLogic performance, be sure to specify a HAVING or WHERE clause to avoid retrieving rows unnecessarily, especially for large tables.

Return Value

GXE.SUCCESS if the method succeeds.

Example 1

```
// Obtain user info from the user database
IQuery query=createQuery();
qry.setTables("CTLcust");
qry.setFields("FirstName, LastName, BillAddr1, BillAddr2,
          BillCity, BillState, BillZip");
qry.setWhere("CustomerId"+"="+cusId+" and
          "+ "Password"+"='"+password+"'");
```

```
IResultSet rs=conn.executeQuery(0, qry, null, null);
if((rs==null)||(rs.getRowNumber()==0))
. . . process result set . . .
```

Example 2

```
// Specify WHERE clause for two different queries
qry1.setWhere("urbanPopulation >= 1000000");
qry2.setWhere("region = 'West' or region = 'East'");
```

Related Topics

createQuery( ) in the AppLogic class (deprecated)

IResultSet Interface *(deprecated)*

IResultSet is deprecated and is provided for backward compatibility only. New applications should use the ResultSet interface from the JDBC Core API.

The IResultSet interface represents the results of a flat query. IResultSet provides methods to iterate through rows in the result set and retrieve data from each row. To retrieve data from the result set, the AppLogic uses methods tailored for specific column types. For example, if retrieving data from a string column, use getValueString( ). If retrieving binary data, use getValueBinary( ).

To process hierarchical result sets, use methods in the IHierResultSet Interface (deprecated) or evalTemplate( ) in the AppLogic class (deprecated) instead.

IResultSet is part of the Data Access Engine (DAE) service.

To create an instance of the IResultSet interface, use executeQuery( ) in the IDataConn Interface (deprecated) or execute( ) in the IPreparedQuery Interface (deprecated), as shown in the following example:

```
IResultSet rs;
rs = conn.executeQuery(0, qry, null, null);
```

Package

com.kivasoft

Methods

Method	Description
close( )	Releases the connection used by the result set.
enumColumnReset( )	Resets the column enumeration to the first column in the result set.
enumColumns( )	Returns the definition of the next column in the result set.
fetchNext( )	Retrieves the next row in the result set.
getColumn( )	Returns the column definition of the column with the specified name.
getColumnByOrd( )	Returns the column definition for the column in the specified ordinal position.
getColumnOrdinal( )	Returns the ordinal position of the column with the specified name.
getNumColumns( )	Returns the number of columns in the result set.
getOrder( )	For asynchronous queries, returns an IOrder object used for obtaining the current status of the query.
getRowNumber( )	Returns the number of the current row in the result set.
getStatus( )	Returns the processing status of the asynchronous database operation on the database server.
getValueBinary( )	Returns the value of a BINARY column from the current row in the result set.
getValueBinaryPiece( )	Returns the value of a LONGBINARY column from the current row in the result set.
getValueDateString( )	Returns the value of a Date type column from the current row in the result set.
getValueDouble( )	Returns the value of a double type column from the current row in the result set.
getValueInt( )	Returns the value of an int type column from the current row in the result set.
getValueSize( )	Returns the cumulutive number of bytes that have been fetched from a column in the current row of the result set.
getValueString( )	Returns the value of a String type column from the current row in the result set.
getValueText( )	Returns the value of a TEXT column from the current row in the result set.

Method	Description
getValueTextPiece( )	Returns the value of a LONGTEXT column from the current row in the result set.
moveTo( )	Moves to the specified row in the result set.
rowCount( )	Returns the total number of rows retrieved thus far from the data source.
wasNull( )	Checks if the value of a column is null or not.

Example

The following code copies the data from all the rows in a result set into a log file:

```
do {
   int invID
   double invTotal;
   String invCustomer;
   Date invDate
   invID = rs.getValueInt(1);
   invDate = rs.getValueDate(2);
   invTotal = rs.getValueDouble(3);
   invCustomer = rs.getValueString(4);
   Log("Invoice   : " + String.valueOf(invID));
   Log(" Date    : " + String.valueOf(invDate));
   Log(" Total    : " + String.valueOf(invTotal));
   Log(" Customer : " + invCustomer);
} while (rs.fetchNext() == 0);
```

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

close( )

Releases the connection used by the result set.

Syntax

```
public int close(
   int dwFlags)
```

dwFlags. Specify 0 (zero). Internal use only.

Usage

Call close( ) to release a connection used by a result set object when the connection is no longer required. An AppLogic should release unused connections to prevent bottlenecks, especially for applications that support many concurrent users, or that access heavily-used databases.

Tip

After calling close( ), release the result set object by calling the GX.Release( ) method.

Return Value

GXE.SUCCESS if the method succeeds.

enumColumnReset( )

Resets the column enumeration to the first column in the result set.

Syntax

```
public int enumColumnReset()
```

Usage

Use enumColumnReset( ) before iterating through and retrieving columns in a result set. The enumColumnReset( ) method ensures that column retrieval starts from the first column.

Thereafter, use enumColumns( ) to retrieve each column sequentially. Each enumColumns( ) call returns an IColumn object for the next column.

Return Value

GXE.SUCCESS if the method succeeds.

Example

The following code enumerates columns in a result set and creates an HTML page that displays the name and type of the columns:

```
String htmlString;

IColumn col;
```

```
IResultSet rs = conn.executeQuery(0, query, null, null);
htmlString += "<h2>Products Table</h2>";
rs.enumColumnReset();
while ((col = rs.enumColumns()) != null) {
   htmlString += "Column Name = ";
   htmlString += col.getName();
   htmlString += ", Column Type = ";
   htmlString += col.getType();
   htmlString += "<br>";
};
return result(htmlString)
```

Related Topics

enumColumns( )

Returns the definition of the next column in the result set.

Syntax

```
public IColumn enumColumns()
```

Usage

Use enumColumns( ) when the column definition is unknown and required for subsequent operations. The AppLogic can use the returned IColumn object to determine characteristics of the column, such as its name, data type, size, whether nulls are allowed, and so on.

Before iterating through columns, the AppLogic should call enumColumnReset( ) to ensure that enumColumns( ) starts with the first column in the table. Each subsequent enumColumns( ) call moves to the next sequential column in the result set and retrieves its column definition in an IColumn object.

Tips

- The columns might not be returned in the order in which they are defined in the database catalog.
- Test for null to determine when the last column has been retrieved.

Return Value

IColumn object containing the next column of data, or null for failure (such as no more columns in the table).

Example

The following code enumerates columns in a result set and creates an HTML page that displays the name and type of the columns:

```
String htmlString;
IColumn col;


IResultSet rs = conn.executeQuery(0, query, null, null);
htmlString += "<h2>Products Table</h2>";
rs.enumColumnReset();
while ((col = rs.enumColumns()) != null) {
   htmlString += "Column Name = ";
   htmlString += col.getName();
   htmlString += ", Column Type = ";
   htmlString += col.getType();
   htmlString += "<br>";
};
return result(htmlString)
```

Related Topics

IColumn Interface (deprecated)

fetchNext( )

Retrieves the next row in the result set.

Syntax

```
public int fetchNext()
```

Usage

Use fetchNext( ) when iterating through rows in the result set to retrieve the contents of the next sequential row and put them in the row buffer for subsequent processing (if RS_BUFFERING has been turned ON).

If result set buffering was activated, fetchNext( ) checks the buffer first before fetching the result set from the actual data source. For more information about result set buffering, see the description of the props parameter of executeQuery( ) in the IDataConn Interface (deprecated).

Tips

- If the AppLogic needs to iterate through the result set more than once, be sure to start with the first row again by calling moveTo( ) and specifying row number 1. This works only when buffering is enabled.
- If result set buffering is enabled, the AppLogic can use moveTo( ) to go to any row in the buffer.

Return Value

GXE.SUCCESS if the method succeeds. The AppLogic can test for the following return values:

- GX_DA_FETCHNEXT_RESULTS.GX_DA_END_OF_FETCH indicates that the end of the result set has been reached.
- GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED indicates that the buffer size has been exceeded. When the buffer is exceeded, subsequent rows are still fetched and still buffered, but GX_DA_FETCHNEXT_RESULTS.GX_DA_BUFFER_EXCEEDED is returned for each overflow row.

Example

The following code copies the data from all the rows in a result set into a log file:

```
do {
   int invID
   double invTotal;
   String invCustomer;
   Date invDate
   invID = rs.getValueInt(1);
   invDate = rs.getValueDate(2);
   invTotal = rs.getValueDouble(3);
   invCustomer = rs.getValueString(4);
   Log("Invoice   : " + String.valueOf(invID));
   Log(" Date    : " + String.valueOf(invDate));
```

```
   Log(" Total    : " + String.valueOf(invTotal));
   Log(" Customer : " + invCustomer);
} while (rs.fetchNext() == 0);
```

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

getColumn( )

Returns the column definition of the column with the specified name.

Syntax

```
public IColumn getColumn(
   String colName)
```

colName. Name of a column or column alias (such as computed columns) in the result set, or an empty string if no alias is specified for the computed column.

Usage

Use getColumn( ) when the data definition of the column is unknown and is required for subsequent operations. The AppLogic can then use methods in the IColumn Interface (deprecated) to obtain descriptive information about a table column from the database catalog, such as the column name, precision, scale, size, table, and data type.

Tips

- Use getColumnByOrd( ) instead when the column position is known but its name is unknown.
- Columns that are the result of query expressions or formulas, such as `invoice.count * product.price`, should have a field alias for the column in the result set. Otherwise, the AppLogic can refer to the column only by its ordinal position. The AppLogic calls setFields( ) in the IQuery Interface (deprecated) to specify field aliases.

Return Value

IColumn object, or null for failure (such as an invalid column name).

Example

```
// Determine whether the column name is valid
IColumn col;
```

```
if((col=rs.getColumn("CustID"))==null)
   return null;
// otherwise, process column using IColumn methods ...
```

Related Topics

IColumn Interface (deprecated)

executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

getColumnByOrd( )

Returns the column definition for the column in the specified ordinal position.

Syntax

```
public IColumn getColumnByOrd(
   int colIndex)
```

colIndex. Ordinal position of a column in the result set. The ordinal position of the first column in the result set is 1, the second column is 2, and so on. The ODBC maximum is 255 columns.

Usage

Use getColumnByOrd( ) when the name of the column is unknown and is required for subsequent operations. The AppLogic can then use methods in the IColumn Interface (deprecated) to obtain descriptive information about a table column from the database catalog, such as the column name, precision, scale, size, table, and data type.

Tip

Use getColumn( ) instead when the column name is known but its ordinal position is unknown.

Return Value

IColumn object, or null for failure (such as an invalid column position).

Example

```
// Determine whether the column position is valid
IColumn col;
if((col=rs.getColumnByOrd(colOrd))==null)
   return null;
```

```
// otherwise, process column using IColumn methods . . .
```

Related Topics

IColumn Interface (deprecated)

executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

getColumnOrdinal( )

Returns the ordinal position of the column with the specified name.

Syntax

```
public int getColumnOrdinal(
   String szColumn)
```

szColumn. Name of a column in the result set.

Usage

Use getColumnOrdinal( ) when the ordinal position of the column is unknown but is required for subsequent operations. For example, the ordinal position of a column is a required parameter value for the getValue**( ) methods, such as getValueString( ) and getValueInt( ).

Return Value

An int value representing the ordinal position of the specified column, or zero for failure (such as an invalid column name). The ordinal position of the first column in the result set is 1, the second column is 2, and so on.

Example

```
// Execute the query
IResultSet rs=conn.executeQuery(0, query, null, null);


// Check for a non empty result
if((rs!=null)&&(rs.getRowNumber()>0))
   return result("Sorry, this user already exists");


// Get the ordinal positions of named columns
int cCustId = rs.getColumnOrdinal("CustomerId");
int cFirst = rs.getColumnOrdinal("FirstName");
```

```
int cLast = rs.getColumnOrdinal("LastName");
int cPassword = rs.getColumnOrdinal("Password");
int cAddr1 = rs.getColumnOrdinal("BillAddr1");
int cAddr2 = rs.getColumnOrdinal("BillAddr2");
int cCity = rs.getColumnOrdinal("BillCity");
int cState = rs.getColumnOrdinal("BillState");
int cZip = rs.getColumnOrdinal("BillZip");
int cEmail = rs.getColumnOrdinal("Email");
int cPhone = rs.getColumnOrdinal("Phone");
if((cCustId*cFirst*cLast*cPassword*cAddr1*cAddr2*cCity*cState*cZip*
cEmail*cPhone)==0)
   return result("Can't map columns");
```

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

getNumColumns( )

Returns the number of columns in the result set.

Syntax

```
public int getNumColumns()
```

Usage

Use getNumColumns( ) if the number of columns in the result set is unknown and required for subsequent operations. For example, when iterating through columns in the result set, the AppLogic can use this information to specify the maximum number of iterations.

Return Value

An int value representing the number of columns, or zero for failure.

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

getOrder( )

For asynchronous queries, returns an IOrder object used for obtaining the current status and return value of the query.

Syntax

```
public IOrder getOrder()
```

Usage

Use getOrder( ) to create an IOrder object that the AppLogic can use to return status information about an asynchronous query.

Rule

The query must be run asynchronously. To run an asynchronous query, the AppLogic must specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC as the dwFlags parameter in executeQuery( ) in the IDataConn Interface (deprecated).

Tips

- The AppLogic can determine the status of the query (active, done, cancelled, or unknown) using getState( ) in the IOrder Interface (deprecated).
- Alternatively, use WaitForOrder( )in theGX class (deprecated), which waits until the asynchronous operation is done, to determine the processing status of an asynchronous query.

Return Value

IOrder object representing the status and return value of the asynchronous query, or null for failure (such as when the query is synchronous).

Related Topics

IOrder Interface (deprecated)

executeQuery( ) in the IDataConn Interface (deprecated)

getRowNumber( )

Returns the number of the current row in the result set.

Syntax

```
public int getRowNumber()
```

Usage
When iterating through rows in the result set, use getRowNumber( ) to keep track of the number of rows processed.

Return Value
An int value representing the number of the current row, or zero for an empty result set or failure (such as an invalid row). The number of the first row in the result set is 1, the second row is 2, and so on. If zero is returned the first time the AppLogic calls getRowNumber( ), that means the result set is empty.

Example
```
// Execute the query

IResultSet rs = conn.executeQuery(0, qry, null, null);

// Test for an empty result set

if((rs==null)||(rs.getRowNumber()==0))

   return result("Customer does not exist");
```

Related Topics
executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

getStatus( )
Returns the processing status of the asynchronous database operation on the database server.

Syntax
```
public int getStatus()
```

Usage
Use getStatus( ) to return status information to use in error-handling code.

Return Value
One of the following:

Value	Description
GXORDER_STATE_ACTIVE	The asynchronous database operation is still being processed.

Value	Description
GXORDER_STATE_CANCEL	The asynchronous database operation has been cancelled.
GXORDER_STATE_DONE	The asynchronous database operation has been completely processed.
GXORDER_STATE_UNKNOWN	The status of the asynchronous database operation is unknown.

getValueBinary( )

Returns the value of a BINARY column in the current row of the result set.

Syntax

```
public byte[] getValueBinary(
   int Ordinal)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

Usage

Use getValueBinary( ) to retrieve binary data of which the total size is equal to or smaller than 64Kb. If the value of the data is larger than 64Kb, use getValueBinaryPiece( ).

Rule

The data type of the column must be BINARY, VARBINARY, or equivalent database type.

Tip

If the value of the data is of type LONGBINARY, use getValueBinaryPiece( ).

Return Value

A byte array for success, or null for failure (such as an invalid column number or data type mismatch).

getValueBinaryPiece( )

Returns the value of a LONGBINARY column in the current row from the result set.

Syntax

```
public byte[] getValueBinaryPiece(
    int Ordinal,
    int nLength)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

nLength. The requested length of the data, in bytes. Up to 64Kb.

Usage

Use getValueBinaryPiece( ) to retrieve binary data of which the total size is larger than 64K. Such binary data must be retrieved in 64K increments. Therefore, you might use getValueBinaryPiece( ) several times to retrieve large amounts of data.

Rules

- The data type of the column must be longvarbinary or equivalent database vendor binary type.
- You cannot call getValueBinaryPiece( ) for a row after you call fetchNext( ).

Tips

- To determine the total size of the binary data that has been retrieved, use getValueSize( ).
- To retrieve binary data of which the total size is less than 64Kb, use getValueBinary( ).

Return Value

A byte array for success, or null for failure (such as an invalid column number or data type mismatch).

Example

The following example shows how to retrieve BLOBs from a database:

```
IQuery     query = null;
IResultSet rs    = null;


query = createQuery();


query.setTables("blobtable");
query.setFields("blobcol");
```

```
rs = conn.executeQuery(0, query, null, null);

if (rs != null && rs.getRowNumber() > 0)
{
   byte   BlobChunk[];
   int expectSize, gotSize;
   expectSize = 65535;

   BlobChunk = rs.getValueBinaryPiece(1, expectSize);
   gotSize = rs.getValueSize(1);

   if (gotSize == expectSize)
      System.out.println("got a full chunk," +
        "size = "+gotSize);
   else
      System.out.println("got a partial chunk," +
        "size = "+gotSize);

   rs.close(0);
}
```

getValueDateString( )

Returns the value of a Date type column, as a string, from the current row in the result set.

Syntax

```
public String getValueDateString(
   int colIndex)
```

colIndex. Ordinal position of a column in the result set. The ordinal position of the first column in the result set is 1, the second column is 2, and so on.

Usage

Use getValueDateString( ) to retrieve date values from the result set for subsequent processing. The following is an example of the format in which getValueDateString( ) returns a date:

```
Jan 26 1998 12:35:00
```

Rule

The specified column must be a Date, Date Time, or Time data type.

Return Value

The date value as a string, or null for failure (such as an invalid column number or data type mismatch).

Example

```
IResultSet rs=conn.executeQuery(0, qry, null, null);
// Check input parameters
if((rs==null)||(rs.getRowNumber()==0)||(colName==null))
   return null;
int colOrd=rs.getColumnOrdinal(colName);
IColumn col;
if((col=rs.getColumnByOrd(colOrd))==null)
   return null;
String valStr=null;
// Switch type
switch(col.getType()) {
   case GX_DA_DATA_TYPES.GX_DA_TYPE_STRING:
      valStr = rs.getValueString(colOrd);
      break;
   case GX_DA_DATA_TYPES.GX_DA_TYPE_LONG:
      valStr=String.valueOf(rs.getValueInt(colOrd));
      break;
   case IGX_DA_DATA_TYPES.GX_DA_TYPE_DATE:
      valStr=rs.getValueDateString(colOrd);
      break;
```

```
   case GX_DA_DATA_TYPES.GX_DA_TYPE_DOUBLE:
      valStr=String.valueOf(rs.getValueDouble(colOrd));
      break;
   default:
};
return valStr;
```

getValueDouble( )

Returns the value of a double type column from the current row in the result set.

Syntax

```
public double getValueDouble(
   int colIndex)
```

colIndex. Ordinal position of a column in the result set. The ordinal position of the first column in the result set is 1, the second column is 2, and so on.

Usage

Use getValueDouble( ) to retrieve decimal, floats, real, numeric, and double values from the result set for subsequent processing.

Rule

The specified column must be a double data type.

Return Value

A double value, or zero for failure (such as an invalid column number or data type mismatch).

Example 1

The following code retrieves data from a result set:

```
// Execute the query
rs = conn.executeQuery(0, query, null, null);
if((rs==null)||(rs.getRowNumber()==0))
{
   // It does not exist, so remove it from the basket
   session.removeProduct(product, this);
   continue;
```

```
}
// Retrieve values from the result set
String prodName=
rs.getValueString(rs.getColumnOrdinal("ProdName"));
double price;
price = rs.getValueDouble(rs.getColumnOrdinal("Price"));
```

Example 2

The following code copies the data from all the rows in a result set into a log file:

```
do {
   int invID
   double invTotal;
   String invCustomer;
   Date invDate
   invID = rs.getValueInt(1);
   invDate = rs.getValueDate(2);
   invTotal = rs.getValueDouble(3);
   invCustomer = rs.getValueString(4);
   Log("Invoice   : " + String.valueOf(invID));
   Log(" Date    : " + String.valueOf(invDate));
   Log(" Total    : " + String.valueOf(invTotal));
   Log(" Customer : " + invCustomer);
} while (rs.fetchNext() == 0);
```

getValueInt( )

Returns the value of an int type column from the current row in the result set.

Syntax

```
public int getValueInt(
   int colIndex)
```

colIndex. Ordinal position of a column. The ordinal position of the first column in the result set is 1, the second column is 2, and so on.

Usage

Use getValueInt( ) to retrieve int or long values from the result set for subsequent processing.

Rule

The specified column must be an int or long data type.

Return Value

An int value, or zero for failure (such as an invalid column number or data type mismatch).

Example 1

The following code retrieves data from a result set:

```
// Execute the query
IResultSet rs=conn.executeQuery(0, query, null, null);
// Check the results
if((rs==null)||(rs.getRowNumber()>0))
{
// Pull the current inventory and name for the product
int prodInv=
rs.getValueInt(rs.getColumnOrdinal("COL_INVENTORY"));
String prodName=
rs.getValueString(rs.getColumnOrdinal("COL_PRODNAME"));
```

Example 2

The following code copies the data from all the rows in a result set into a log file:

```
do {
   int invID
   double invTotal;
   String invCustomer;
   Date invDate
   invID = rs.getValueInt(1);
   invDate = rs.getValueDate(2);
   invTotal = rs.getValueDouble(3);
   invCustomer = rs.getValueString(4);
```

```
   Log("Invoice   : " + String.valueOf(invID));
   Log(" Date    : " + String.valueOf(invDate));
   Log(" Total    : " + String.valueOf(invTotal));
   Log(" Customer : " + invCustomer);
} while (rs.fetchNext() == 0);
```

getValueSize( )

Returns the cumulutive number of bytes that have been fetched from a column in the current row of the result set.

Syntax

```
public int getValueSize(
   int colIndex);
```

colIndex. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

Usage

Use getValueSize( ) during data retrieval to check the size of the BLOB column that has been retrieved. When the AppLogic first calls getValueSize( ) before calling getValueBinaryPiece( ) to retrieve the value of a LONGBINARY column, getValueSize( ) returns 0.

Each subsequent getValueSize( ) call during data retrieval returns the cumulative size of the data that has been retrieved.

Return Value

An integer value representing the number of bytes retrieved from a column, or a negative number for failure.

Related Topics

getValueBinaryPiece( )

getValueString( )

Returns the value of a String type column from the current row in the result set.

Syntax

```
public String getValueString(
   int colIndex)
```

colIndex. Ordinal position of a column in the result set. The ordinal position of the first column in the result set is 1, the second column is 2, and so on.

Usage

Use getValueString( ) to retrieve String values from the result set for subsequent processing.

Rule

The specified column must be a String data type.

Return Value

A String value, or null for failure (such as an invalid column number or data type mismatch).

Example 1

The following code retrieves data from a result set:

```
// Execute the query
rs = conn.executeQuery(0, query, null, null);
if((rs==null)||(rs.getRowNumber()==0))
{
   // It does not exist, so remove it from the basket
   session.removeProduct(product, this);
   continue;
}
// Retrieve values from the result set
String prodName=
rs.getValueString(rs.getColumnOrdinal("ProdName"));
double price=
rs.getValueDouble(rs.getColumnOrdinal("Price"));
```

Example 2

The following code copies the data from all the rows in a result set into a log file:

```
do {
   int invID
   double invTotal;
   String invCustomer;
```

```
    Date invDate
    invID = rs.getValueInt(1);
    invDate = rs.getValueDate(2);
    invTotal = rs.getValueDouble(3);
    invCustomer = rs.getValueString(4);
    Log("Invoice   : " + String.valueOf(invID));
    Log(" Date    : " + String.valueOf(invDate));
    Log(" Total    : " + String.valueOf(invTotal));
    Log(" Customer : " + invCustomer);
} while (rs.fetchNext() == 0);
```

getValueText( )

Returns the value of a TEXT column in the current row from the result set.

Syntax

```
public String getValueText(
    int Ordinal)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

Usage

Use getValueText( ) to retrieve TEXT data of which the total size is equal to or smaller than 64K.

Rule

The data type of the column must be TEXT or database equivalent.

Tips

- To determine the actual size of the TEXT data, use getValueSize( ).
- If the value of the data is of type LONGTEXT, use getValueTextPiece( ).

Return Value

A String object, or null for failure (such as an invalid column number or data type mismatch).

getValueTextPiece( )

Returns the value of a LONGTEXT column in the current row from the result set.

Syntax

```
public String getValueTextPiece(
    int Ordinal,
    int nLength)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

nLength. The requested length of the data, in bytes. Up to 64Kb.

Usage

Use getValueTextPiece( ) to retrieve LONGTEXT data. LONGTEXT values must be retrieved in 64K increments, therefore, you must use getValueTextPiece( ) repeatedly to retrieve the data.

Rules

- The data type of the column must be LONGTEXT or database equivalent.
- Call getValueTextPiece( ) until you get all the data before calling fetchNext( ) again.

Tips

- To determine the actual size of the LONGTEXT data, use getValueSize( ). The actual size of the data determines the number of times you need to call getValueTextPiece( ).
- For data of type TEXT, use getValueText( ).

Return Value

A String object, or null for failure (such as an invalid column number or data type mismatch).

moveTo( )

Moves to the specified row in the result set.

Syntax

```
public int moveTo(
    int nRow)
```

nRow. Number of the row in the result set to move to. The number of the first row in the result set is 1, the second row is 2, and so on.

Usage

Use moveTo( ) to move the internal cursor to a specific row in the result set, skipping over rows to be excluded from processing. In addition, if RS_BUFFERING is ON, after iterating through all rows in a result set, the AppLogic can return to the first row in the result set in preparation for the next iteration.

Rules

- The specified row number must exist in the result set.
- If row buffering is not enabled for the result set, the AppLogic can move forward to subsequent rows only. The AppLogic cannot return to rows that have been processed previously.

Tip

Use rowCount( ), if the database driver supports it, to obtain the maximum number of rows in the result set.

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

rowCount( )

Returns the total number of rows retrieved thus far from the data source.

Syntax

```
public int rowCount()
```

Usage

Use rowCount( ) to return the current number of rows processed so far in the result set. This method is useful for checking that data exists in the result set before processing the result set.

If iterating through rows in a result set that has been completely returned, use rowCount( ) to determine the current maximum number of rows to process.

Tip

If result set buffering is enabled, the AppLogic can use rowCount( ) to find the current number of rows in the buffer.

Return Value

An int value representing the total number of rows in the result set, or zero for failure (such as an empty result set or unable to retrieve any rows).

Example

```
do {
   htmlString += rs.getValueString(DESCRIPTION_COL);


   // Truncate the report after 10 rows.
   if (rs.rowCount() > 10)
      break;
} while (rs.fetchNext() == GXE_SUCCESS);
```

Related Topics

executeQuery( ) in the IDataConn Interface (deprecated)

execute( ) in the IPreparedQuery Interface (deprecated)

wasNull( )

Checks if the value of a column is null or not.

Syntax

```
public boolean wasNull(
   int Ordinal)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

Usage

Use wasNull( ) to check if a column value is null or not. This method is useful for determining if a null return value is an error condition or if the column contained no value.

Return Value

True if the column value is null, false if the value is not.

IRowSet2 Interface

IRowSet2 is an extension to the javax.sql.RowSet interface. IRowSet2 includes a set of methods that are needed for some of the server's backend. In particular, this interface includes initialization methods and some "set" methods that associate a RowSet to a particular servlet request or servlet response.

Although anyone developing a iPlanet Application Server application can use IRowSet2, this interface is intended for use in components generated by Netscape Application Builder.

IRowSet2 replaces the IDataSet interface from iAB 3.0.

Package

com.netscape.server.servlet.extension

Methods

Method	Description
getName( )	Retrieves the name under which the RowSet is stored in the HttpServletRequest object.
init( )	Initializes the RowSet for execution.
initMetaInfo( )	Initializes the RowSet to match specific metadata.
setName( )	Sets the name of the RowSet.
setRequest( )	Sets the request for this RowSet.
setResponse( )	Sets the response for this RowSet.

Related Topics

javax.sql.RowSet interface

getName( )

Retrieves the name under which the RowSet is stored in the HttpServletRequest object.

Syntax

```
public abstract String getName()
```

Usage

Use getName( ) to retrieve the name of a RowSet. If the RowSet is built over the ITemplateData interface, then the retrieved name is also the group name.

Related Topics

setName( ),
setAttribute( ) in class javax.servlet.http.HttpServletRequest,
ITemplateData interface

init( )

Initializes the RowSet for execution.

Syntax

```
public abstract void init()
```

Usage

Use init( ) to initialize the RowSet. For example, you can call init( ) to load a query from a file.

Related Topics

initMetaInfo( ),
DBRowSet class

initMetaInfo( )

Initializes the RowSet to match specific metadata.

Syntax

```
public boolean initMetaInfo(
   OrderedHash metaData) throws SQLException
```

metaData. An OrderedHash of NTV properties. The RowSet will be initialized with these properties.

Usage

Use initMetaInfo( ) to initialize a RowSet. This method is called by BaseUtils.initRowSets( ) when the RowSet is created.

The metaData parameter specifies the properties stored in a method in the NTV list. The properties specify the query file name, the query name, the connection name, and the input bindings for the query parameters. In this way, the RowSet is initialized so as to match the properties stored in the NTV-list method.

The initMetaInfo( ) method must be called when the RowSet is constructed; that is, before init( ). For example, you can call initMetaInfo( ) to set the filename and query for a subsequent query-load.

Return Value

Returns false if initialization fails.

setName( )

Sets the name of the RowSet.

Syntax

```
public abstract void setName(
   String newName)
```

newName. The name you want to call this RowSet.

Usage

Use setName( ) to set the name of the RowSet. This method must be called when the RowSet is constructed. You can also use getName( ) to retrieve the name of the RowSet.

Note that the setName( ) method does not store the RowSet into the request under the name specified in this call.

Related Topics

getName( )

setRequest( )

Sets the request for this RowSet.

Syntax

```
public abstract void setRequest(
   HttpServletRequest request)
```

request. The HttpServletRequest object.

Usage

Use setRequest( ) and setResponse( ) to set the request and response objects for this RowSet. These methods must be called when the RowSet is constructed. Calling these methods will associate the RowSet with a particular servlet request or servlet response.

Related Topics
setResponse( )

setResponse( )
Sets the response for this RowSet.

Syntax
```
public abstract void setResponse(
   HttpServletResponse response)
```

response. The HttpServletResponse object.

Usage
Use setRequest( ) and setResponse( ) to set the request and response objects for this RowSet. These methods must be called when the RowSet is constructed. Calling these methods will associate the RowSet with a particular servlet request or servlet response.

Related Topics
setRequest( )

ISequence Interface *(deprecated)*
ISequence is deprecated and is provided for backward compatibility only.

The ISequence interface represents a sequence in an underlying database.

Sequences are implemented in the database server to provide unique, incremental numbers assigned to records in a database. For example, the AppLogic can create a customer ID sequence to generate customer IDs, or create a purchase order sequence to generate purchase order numbers.

The ISequence interface provides methods to determine the current sequence value or to increment to the next sequence value. Sequences are useful for many types of applications, such as order entry applications.

The ISequence interface is part of the Data Access Engine (DAE) service.

To create an instance of the ISequence interface , use createSequence( ) in the ISequenceMgr Interface (deprecated), as shown in the following example:

```
IDataConn dc;

dc = createDataConn(...);
```

```
ISequenceMgr sm;
sm = (ISequenceMgr) dc;
ISequence seq;
seq = sm.createSequence(...);
```

Package

com.kivasoft

Methods

Method	Description
drop( )	Deletes the sequence from the database.
getCurrent( )	Returns the current value in the sequence.
getNext( )	Increments the sequence and returns its incremented value.

Related Topics

createSequence( ) in the ISequenceMgr Interface (deprecated)

drop( )

Deletes the sequence from the database.

Syntax

```
public int drop();
```

Usage

Use drop( )to remove a sequence from the database. Be careful when using this method. If the database implements the sequence as a field in a table, drop( ) will delete the entire table, not just the sequence field. If the database implements the sequence as an object, as does Oracle for example, drop( ) deletes only the sequence object.

Typically, once you start a sequence there is no reason to delete it. The sequence is normally used to create a permanent, unique numbering system for data in a database. However, you might use drop( ) if you are using the sequence mechanism to generate unique sequential numbers for a temporary programmatic purpose.

Return Value
GXE.SUCCESS if the method succeeds

Related Topics
createSequence( ) in the ISequenceMgr Interface (deprecated)

getCurrent( )
Returns the current value in the sequence.

Syntax
```
public int getCurrent();
```

Usage
Use getCurrent( ) to obtain the current value of the sequence without actually incrementing the sequence value.

Alternatively, use getNext( ) to increment the sequence and obtain its incremented value.

Rule
For Oracle databases, the session must first call getNext( ) before it can call getCurrent( ).

Tip
UnlikegetNext( ) , calling getCurrent( ) does not change the value of the sequence.

Return Value
An int value representing the current value of the sequence, or -1 for failure.

Related Topics
createSequence( ) in the ISequenceMgr Interface (deprecated)

getNext( )
Increments the sequence and returns its incremented value.

Syntax
```
public int getNext();
```

Usage

Use getNext( ) to increment and return the value of the sequence by the amount specified in the increment parameter in the createSequence( ) method in the ISequenceMgr Interface (deprecated). The incrementation value is always a positive integer.

Alternatively, use getCurrent( ) to obtain the current value of the sequence without incrementing the sequence.

Return Value

An int value representing the next (incremente)d value of the sequence, or -1 for failure.

Rules

- For Informix and Sybase databases, the session that creates the sequence must call getNext( ) at least once before any other session can call getSequence( ) in the ISequenceMgr Interface (deprecated).
- For Oracle databases, the session must first call getNext( ) before it can call getCurrent( ).

Tip

Successive calls to getNext( ) return successive integers.

Example

```
IDataConn conn;

conn = createDataConn(0, GX_DA_DAD_DRIVERS.GX_DA_DRIVER_ODBC,
   "salesDB", "salesDB", "steve", "pass7878");


// Cast the connection to an ISequenceMgr interface

// and set up the sequence

ISequence seq;

seq = ((ISequenceMgr) conn).createSequence("mySeq", "orders.ID",
100, 1, null)


// To start the sequence, call getNext().

int seqVal;

seqVal = seq.getNext();
```

```
// Use the sequence number to perform the task for
// which you created it.
IQuery qry = createQuery();
qry.setSQL("INSERT into orders (ID) values ("+ seqVal +")," +
  "(cust) values ("+ custname +")");
```

Related Topics

createSequence( ) in the ISequenceMgr Interface (deprecated)

ISequenceMgr Interface *(deprecated)*

ISequenceMgr is deprecated and is provided for backward compatibility only.

The ISequenceMgr interface provides methods for creating and retrieving an ISequence object, which represents a sequence in an underlying database. Sequences provide unique, incremental numbers assigned to records in a database. After creating a sequence by calling createSequence( ), the AppLogic can use methods in the ISequence Interface (deprecated) to retrieve sequence values.

The ISequenceMgr interface is part of the Data Access Engine (DAE) service.

The ISequenceMgr interface is implemented by the IDataConn object. To use it, cast IDataConn to the ISequenceMgr interface, as shown in the following example:

```
IDataConn dc;
dc = createDataConn(...);
ISequenceMgr sm;
sm = (ISequenceMgr) dc;
```

Package

com.kivasoft

Methods

Method	Description
createSequence( )	Creates a new sequence object in the underlying database.
getSequence( )	Returns an existing sequence object for the specified sequence name in the underlying database.

Related Topics

ISequence Interface (deprecated)

createSequence( )

Creates a new sequence object in the underlying database.

Syntax

```
public ISequence createSequence(
   String szName,
   String szCol,
   int dwStart,
   int dwIncrement,
   String szOptions);
```

szName. Name of the sequence. The name can be simple (such as "mySeq") or qualified with the name of the database owner (such as "mary.mySeq").

szCol. Name of the column in the database table to use if the database supports sequence column types. For more information, see your database vendor's documentation. If null, defaults to "SEQVAL".

dwStart. Starting value of the sequence. Must be a positive integer.

dwIncrement. Value by which to increment the sequence with each call to getNext( ). Must be a positive integer. Defaults to one (1). Not all databases support this feature. For more information, see your database vendor's documentation.

szOptions. Additional sequence creation options that are database vendor-specific:

- For Oracle, these are options to the "CREATE Sequence"command.
- For SQL Server (Sybase and Microsoft) databases, these are column options for the "CREATE Table"command.
- For Informix, no options exist.

For more information, see your database vendor's documentation.

Usage

Use createSequence( ) to create a new ISequence object, representing an incremental number generator, with the specified starting value. The AppLogic can then use methods in the ISequence Interface (deprecated) to obtain the current or next value of this sequence object.

Sequences provide unique, incremental numbers assigned to records in a database. For example, you can create a customer ID sequence to generate customer IDs, or create a purchase order sequence to generate purchase order numbers.

Tip

For Oracle databases, createSequence( ) creates a sequence object. For Sybase, Informix, and Microsoft SQL Server databases, createSequence( ) creates a table object with a sequence column.

Return Value

ISequence object, or null for failure (such as an invalid database connection).

Example

```
IDataConn conn;

conn = createDataConn(0, GX_DA_DAD_DRIVERS.GX_DA_DRIVER_ODBC,

   "salesDB", "salesDB", "steve", "pass7878");


// Cast the connection to an ISequenceMgr interface

// and set up the sequence

ISequence seq;

seq = ((ISequenceMgr) conn).createSequence("mySeq", "orders.ID",
100, 1, null)


// To start the sequence, call getNext().

int seqVal;

seqVal = seq.getNext();


// Use the sequence number to perform the task for

// which you created it.

IQuery qry = createQuery();

qry.setSQL("INSERT into orders (ID) values ("+ seqVal +")," +

  "(cust) values ("+ custname +")");
```

Related Topics

ISequence Interface (deprecated)

getSequence( )

Returns an existing sequence object, for the specified sequence name, from the underlying database.

Syntax

szName. Name of the sequence. The name can be simple (such as `"mySeq"`) or qualified with the name of the database owner (such as `"mary.mySeq"`).

szCol. Name of the column in the database table to use if the database supports sequence column types. For more information, see your database vendor's documentation. If null, defaults to `"SEQVAL"`.

Usage

Use getSequence( ) to obtain the ISequence object with the specified name in the underlying database. The AppLogic can then use methods in the ISequence Interface (deprecated) to obtain the current or next value of this sequence object.

Sequences provide unique, incremental numbers assigned to records in a database. For example, you can create a customer ID sequence to generate customer IDs, or create a purchase order sequence to generate purchase order numbers.

Rules

- Use createSequence( ) to create the ISequence object.
- The specified sequence name must be valid.
- For Informix and Sybase databases, the session that creates the sequence must call getNext( ) at least once before any other session can call getSequence( ).

Return Value

ISequence object, or null for failure (such as an invalid sequence name).

Related Topics

ISequence Interface (deprecated)iAS

IServerContext Interface

The IServerContext interface provides a way to access iAS-specific server interfaces from within servlets or EJBs. From servlets, the standard servlet context can be cast to an IServerContext object. From EJBs, a javax.ejb.EJBContext object can be cast to an IServerContext.

IServerContext is often used together with the ICallerContext interface.

Package

com.netscape.server

Methods

Method	Description
createIdentityByString( )	Creates an instance of java.security.Identity.
getCallerContext( )	Retrieves the com.netscape.server.ICallerContext object.
getContext( )	Retrieves the com.kivasoft.IContext object.

Related Topics

com.kivasoft.applogic.AppLogic class (deprecated),
com.kivasoft.dlm.GXContext class,
com.kivasoft.IContext Interface,
com.netscape.server.ICallerContext Interface,

javax.ejb.EJBContext interface

createIdentityByString( )

Creates an instance of java.security.Identity.

Syntax

```
public java.security.Identity createIdentityByString(
   String identity)
```

identity. The name to associate with the Identity object.

Usage

Neither of the specifications for servlets and EJBs describe a standard way of creating an instance of java.security.Identity. The createIdentityByString( ) method overcomes this limitation. Use this method to create an Identity instance within a iPlanet Application Server 6.0 application.

Once you create the Identity, you can then (for example) pass it to a subsequent test for equality or use it as the role parameter of the isCallerInRole( ) method. The isCallerInRole( ) method is available in both javax.ejb.EJBContext and com.netscape.server.ICallerContext.

Return Value
A java.security.Identity object with the given name.

getCallerContext( )
Retrieves the com.netscape.server.ICallerContext object.

Syntax
```
public ICallerContext getCallerContext()
```

Usage
Use this method within servlets to identify the caller context.

Currently, the Servlet API specification provides no way to obtain the caller's principal from within a servlet. By contrast, the EJB specification defines the javax.ejb.EJBContext.getCallerPrincipal( ) method.

The getCallerContext( ) method is provided to add support for programmatic security from servlets. However, after this support is added to the Servlet API specification, the getCallerContext( ) method will be deprecated.

Return Value
The ICallerContext object that identifies the caller context within the servlet.

getContext( )
Retrieves the com.kivasoft.IContext object.

Syntax
```
public com.kivasoft.IContext getContext()
```

Usage
Use this method within servlets or EJBs. The getContext( ) method lets you access iAS-specific application modules or extensions that run in a pre-iAS-6.0 environment.

The retrieved IContext object is the old iPlanet Application Server 2.x style context. The getContext( ) method is especially useful when combined with the functions from the com.kivasoft.dlm.GXContext class.

Example
Assume you have declared the following bean member variable containing SessionContext:

```
public transient javax.ejb.SessionContext m_ctx = null;
```

The following sample code shows a bean implementation method that needs to invoke some iPlanet Application Server 2.x functionality:

```
public invokeKivaMethod()
{
    // obtain 2.x style KIVA context from standard bean context

    com.netscape.server.IServerContext sc;
    // cast the standard servlet context or bean context
    //    to IServerContext
    sc = (com.netscape.server.IServerContext) m_ctx;

    // acquire 2.x style context
    com.kivasoft.IContext gxContext = sc.getContext();

    // Use the context with GXContext class to access 2.x
functionality.
    // In this case, choose a logging operation
    com.kivasoft.dlm.GXContext.Log(
          gxContext,
          com.kivasoft.types.GXLOG.GXEVENTTYPE_INFORMATION,
          -1,
          "Send this informational message to server log");

    ...
```

Return Value

The IContext object that identifies the context within the servlet or EJB.

IServletErrorHandler Interface

The interface com.netscape.server.servlet.extension.HttpServletRequest2 uses IServletErrorHandler to inform the user of validation errors or other types of errors.

Although anyone developing an application for iPlanet Application Server can use IServletErrorHandler, this interface is typically used in components generated by Netscape Application Builder.

Package

com.netscape.server.servlet.extension

Methods

Method	Description
getBytes( )	Performs error handling for input parameter validation.
handleSessionVariableError( )	Performs error handling for session validation.
streamError( )	Sends a standard error as a data stream.

handleInputValueError( )

Performs error handling for input parameter validation.

Syntax

```
public abstract int handleInputValueError(
    HttpServletRequest request,
    String name,
    int errorType)
```

request. The HttpServletRequest containing the invalid input.

name. The parameter name that failed validation.

errorType. An integer value identifying the error type. The error type is passed in by the HttpServletRequest2.validate( ) method. Values for possible error types are listed under the validate( ) method.

Usage

Typically, the handleInputValueError( ) method is never called directly by the application programmer. Instead, this method is called by HttpServletRequest2.validate( ) for each error that is found during input parameter validation.

handleInputValueError( ) is called if a parameter fails its metadata-driven validation. You can use this method in one of several ways:

- Use the default handling. As an option, you can call getServletErrorHandler( ) on the request and delegate to the retrieved error handler instead.
- Use the default handling, but change the template to be streamed. This case is presented in Examples.
- Handle the error completely by yourself. This case is presented in Examples.

The default behavior of handleInputValueError( ) is to add values into three validation vectors. These vectors are listed in three methods of HttpServletRequest2: getErrorCodes( ), getErrorMsgs( ), and getErrorVars( ).

To override the default behavior, implement IServletErrorHandler and implement handleInputValueError( ) to handle errors related to input parameter validation. Before calling validate( ), call setServletErrorHandler( ) on the HttpServletRequest2 object.

Tip

It is recommended that you standardize error handling, so you may want to place error handling functionality in the default BaseServlet.

Return Value

This method returns any of the following integer values:

Value	When Is This Returned
SUCCESS	When everything works.
FAIL_STREAM_ERROR or any positive integer	When an error has occurred but no error has yet been streamed to the user; the caller of the method must stream an error. For example, the default implementation returns ERROR_INPUT (a value of 3).
FAIL_DONT_STREAM_ERROR or any negative integer	When an error has occurred and an error has been streamed to the user; the caller of the method should not stream anymore information.

Example

The following code fragment uses the default handling but changes the template that is streamed. Typically, you would create this code in an inner class of the servlet that uses the error handler.

```
public class foo implements IServletErrorHandler
{
```

```
// The old error handler
    IServletErrorHandler m_old

// Store the old servlet error handler somewhere
    public foo (HttpServletRequest2 request)
    {
        m_old = request.getServletErrorHandler();
    }

// Handle the error as before, but set additional information
    public int handleInputValueError(
            HttpServletRequest request,
            String name,
            int errorType)
    {
        m_old.handleInputValueError(request, name, errorType);
        request.setAttribute("AddlInfo", "name failed");
        return FAIL_STREAM_ERROR;
    }

// Not interested in session variables,
// so just delegate to the old handler
    public int handleSessionVariableError(
            HttpServletRequest request,
            String name,
            int errorType)
    {
        return m_old.handleSessionVariableError(request, name,
errorType);
    }
```

```
// Handle the validation error appropriately
   public int streamError(
          HttpServletRequest request,
          HttpServletResponse response,
          Object errorSrc,
          int errorType)
   {
     if (errorType == ERROR_VALIDATION_FAILED)
     {
         BaseUtils.includeJSP(request, response,
"MyErrorTemplate.jsp");
         return FAIL_DONT_STREAM_ERROR;
     }
     else
     {
          // handle the other errors by delegation
          return
m_old.streamError(request,response,errorSrc,errorType)
   }
}
```

In your servlet, you call the validate( ) method after setting the error handler that validate( ) will use when it reports errors:

```
...
IServletErrorHandler handler = new
foo((HttpServletRequest2)request);
((HttpServletRequest2)request).setServletErrorHandler(handler);
...
request.validate(response)
```

Related Topics

handleSessionVariableError( ),
HttpServletRequest2 interface

handleSessionVariableError( )

Performs error handling for session validation.

Syntax

```
public abstract int handleSessionVariableError(
   HttpServletRequest request,
   String name,
   int errorType)
```

request. The HttpServletRequest containing the invalid session.

name. The parameter name that failed validation.

errorType. Unused.

Usage

Typically, the handleSessionVariableError( ) method is never called directly by the application programmer. Instead, this method is called by HttpServletRequest2.validate( ) for each error that is found during session validation.

handleSessionVariableError( ) is called if a session variable fails its metadata-driven validation. You can use this method in one of several ways:

- Use the default handling. As an option, you can call getServletErrorHandler( ) on the request and delegate to the retrieved error handler instead.
- Use the default handling, but change the template to be streamed.
- Handle the error completely by yourself.

The default behavior of handleSessionVariableError( ) is to add values into three validation vectors. These vectors are listed in three methods of HttpServletRequest2: getErrorCodes( ), getErrorMsgs( ), and getErrorVars( ).

To override the default behavior, implement IServletErrorHandler and implement handleSessionVariableError( ) to handle errors related to session validation. Before calling validate( ), call setServletErrorHandler( ) on the HttpServletRequest2 object.

Tip

It is recommended that you standardize error handling, so you may want to place error handling functionality in the default BaseServlet.

Return Value

This method returns any of the following integer values:

Value	When Is This Returned
SUCCESS	When everything works.
FAIL_STREAM_ERROR or any positive integer	When an error has occurred but no error has yet been streamed to the user; the caller of the method must stream an error. For example, the default implementation returns ERROR_SESSION (a value of 2).
FAIL_DONT_STREAM_ERROR or any negative integer	When an error has occurred and an error has been streamed to the user; the caller of the method should not stream anymore information.

Examples

See handleInputValueError( ) for examples.

Related Topics

handleInputValueError( ),
HttpServletRequest2 interface

streamError( )

Sends a standard error as a data stream.

Syntax

```
public abstract int streamError(
   HttpServletRequest request,
   HttpServletResponse response,
   Object errorSource,
   int errorType) throws IOException, ServletException
```

request. The HttpServletRequest object.

response. The HttpServletResponse object.

errorSource. The object containing information about the source of the error.

errorType. An integer value identifying the error type to report. Specify one of the following values:

Error Code	Value	Action
ERROR_NO_SUCH_TEMPLATE	0	For this errorType, set errorSource to the name of the template that could not be found.
ERROR_EXCEPTION_THROWN	1	For this errorType, set errorSource to the exception that was thrown.
ERROR_VALIDATION_FAILED	2	For this errorType, the errorSource is unspecified.
ERROR_TEMPLATE_FAILED	3	For this errorType, set errorSource to the name of the template that failed.

Usage

Typically, the streamError( ) method is called directly by the application programmer as well as by HttpServletRequest2.validate( ) for validation errors. In particular, only the variants of validate( ) that stream an error will call the streamError( ) method. This applies to the two variants of validate( ) that take an HttpServletResponse object.

In addition, streamError( ) may be called by BaseUtils.initRowSets( ) when an exception occurs, or streamError( ) may be called by programmer-defined code.

To override the default behavior, implement IServletErrorHandler and implement streamError( ) to define the template or exception to stream when a validation error (or other type of error) occurs. Before calling validate( ), call setServletErrorHandler( ) on the HttpServletRequest2 object.

Return Value

This method returns any of the following integer values. By default, FAIL_DONT_STREAM_ERROR is returned.

Value	When Is This Returned
SUCCESS	When everything works.
FAIL_STREAM_ERROR	When an error has occurred but no error has yet been streamed to the user; the caller of the method must stream an error.
FAIL_DONT_STREAM_ERROR	When an error has occurred and an error has been streamed to the user; the caller of the method should not stream anymore information.

Related Topics
HttpServletRequest2 interface

ISession2 Interface *(deprecated)*

The ISession2 interface is deprecated and is provided for backward compatibility only. New applications should use the methods provided by the standard javax.servlet.http.HttpSession interface. In addition, iPlanet Application Server provides the HttpSession2 interface, an extension to HttpSession that supports applications that must call AppLogics.

For more information, see the *Programmer's Guide (Java)* or the *Migration Guide*.

The ISession2 interface represents a session between a user and an application.

AppLogics use sessions to store information about each user's interaction with the application. For example, a login AppLogic might create a session object to store the user's login name and password. This session data is then available to other AppLogics in the application.

Session data is stored in a distributed state layer in the iPlanet Application Server, so that the data is available even when the server destroys the AppLogic when it has finished executing. Storing the session data in the distributed state layer also enables AppLogics running in different clusters or servers to access the data.

A session has the following attributes, which are set when the AppLogic creates a session:

- A unique ID. You can specify an ID, or use the default ID the system generates.
- An association with an application. This setting enables the iPlanet Application Server to determine which AppLogics have access to the session data.
- A timeout value. You can specify if the session is automatically destroyed after a specified time. If you don't specify a timeout value (timeout = 0), the session is destroyed when you call the destroySession( ) method in the AppLogic class.
- Scope. You can specify if the session data is available at the local, cluster, or enterprise-wide level.

The ISession2 interface defines methods for setting and retrieving data in a session. It also defines methods for retrieving the attributes—ID, associated application, timeout value, and scope—of a session.

To create an instance of the ISession2 interface, use the createSession( ) method in the AppLogic class.

If your application requires a custom session object, for example, to support additional methods, you can subclass the Session2 class (deprecated) and define your own methods.

Package

com.kivasoft

Methods

Method	Description
getSessionApp( )	Returns the name of the application associated with the session.
getSessionData( )	Returns session data.
getSessionFlags( )	Returns the flags associated with the session when it was created.
getSessionID( )	Returns the session ID.
getSessionTimeout( )	Returns the session's timeout value in seconds.
saveSession( )	Saves changes to a session.
setSessionData( )	Sets session data.

getSessionApp( )

Returns the name of the application associated with the session.

Syntax

```
public String getSessionApp()
```

Usage

Use getSessionApp( ) to retrieve the name of the application associated with the session. All AppLogics in an application have access to the same session data.

Return Value

A string representing the application name, or null for failure.

Example

The following code shows how to create a session and get the name of the application associated with the session:

```
ISession2 sess;
```

```
//Create a session and associate it with myApp application
sess=createSession(GXSESSION.GXSESSION_DISTRIB,
                   0, "myApp", null, null);

//Get the application name associated with the session
//getSessionApp() should return "myApp"
String appname;
appname = sess.getSessionApp();
if (appname == null)
   return result("getSessionApp returned error");
log("Session application name:" + appname);
```

Related Topics
createSession( ) in the AppLogic class

getSessionData( )

Returns session data.

Syntax
```
public IValList getSessionData()
```

Usage
Use getSessionData( ) to retrieve session data for processing. Data is returned in an IValList object. This method retrieves the contents that were last saved in the distributed store with saveSession( ). Use methods in the IValList interface to iterate through and access items in the IValList object.

Return Value
IValList object containing the session information, or null for failure.

Example
```
public class OBSession extends Object
{
   protected ISession2 systemSession;
   public    IValList  sessionValList = null;
```

```
public OBSession(ISession2 app_session)
{
   systemSession = app_session;

   // Retrieve session data
   sessionValList = systemSession.getSessionData();
}
// Method for retrieving the user name from session data
public String getUserName()
{
   // Return the userName value
   return sessionValList.getValString("userName");
```

Related Topics
createSession( ) in the AppLogic class

getSessionFlags( )

Returns the flags associated with the session when it was created.

Syntax
```
public int getSessionFlags()
```

Usage
Use getSessionFlags( ) to retrieve the flags that were specified when the session was created with createSession( ). The following table describes the valid session flags, which are static variables in the GXSESSION class:

Flag	Description
GXSESSION_LOCAL	The session is visible to the local process only.
GXSESSION_CLUSTER	The session is visible to all AppLogics within the cluster.
GXSESSION_DISTRIB	The session is visible to all AppLogics in the enterprise environment.

Flag	Description
GXSESSION_PERSISTENT	The session persists in the event of a server crash.
GXSESSION_TIMEOUT_ABSOLUTE	The session expires at a specific date and time.
GXSESSION_TIMEOUT_CREATE	The session expires *n* seconds from the time the node was created.

Return Value

The flags listed in the previous table.

Example

The following code shows how to create a session and get the associated flags:

```
ISession2 sess;
//Create a session with distributed scope
sess=createSession(GXSESSION.GXSESSION_DISTRIB,
                   0, "myApp", null, null);

//Get the flag assigned to the session
//getSessionFlag() should return GXSESSION_DISTRIB
int flag;
flag = sess.getSessionFlags();
log("Session flag:" + flag);
```

Related Topics

createSession( ) in the AppLogic class

getSessionID( )

Returns the session ID.

Syntax

```
public String getSessionID()
```

Usage

Use getSessionID( ) to retrieve the unique ID associated with a session.

Return Value
A string representing the session ID, or null for failure.

Example
The following code shows how to create a session and get the session ID:

```
ISession2 sess;
//Create a session using the default ID generator
sess=createSession(GXSESSION.GXSESSION_DISTRIB,
                   0, "myApp", null, null);
//Get the session ID
String sessid;
sessid = sess.getSessionID();
if (sessid == null)
   return result("getSessionID returned error");
log("Session ID:" + sessid);
```

Related Topics
createSession( ) in the AppLogic class

getSessionTimeout( )
Returns the session's timeout value in seconds.

Syntax
```
public int getSessionTimeout()
```

Usage
Use getSessionTimeout( ) to find out if a session is terminated after a specified time, or if it needs to be terminated explicitly. A timeout value of 0 means the session ends when it is explicitly terminated with the destroySession( ) method.

Return Value
Integer representing the timeout value in seconds.

Example
The following code shows how to create a session and get session's timeout value:

```
ISession2 sess;
//Create a session with no timeout value
```

```
sess=createSession(GXSESSION.GXSESSION_DISTRIB,
                   0, "myApp", null, null);


//Get the timeout value
//getSessionTimeout() should return 0
int timeout;
timeout = sess.getSessionTimeout();
log("Session timeout value:" + timeout);
```

Related Topics

createSession( ) in the AppLogic class

saveSession( )

Saves changes to a session.

Syntax

```
public int saveSession(
   int dwFlags)
```

dwFlags. Specify 0 (zero).

Usage

Use saveSession( ) to ensure changes are saved in the distributed state storage area, which stores the session information for subsequent use if any other AppLogic objects are invoked within the same session.

Tips

- The AppLogic needs to call the saveSession( ) method in the AppLogic class at least once to set a cookie, which passes the session ID between the Web browser and iPlanet Application Server. The saveSession( ) method in the ISession2 interface only saves data to the distributed state store, whereas saveSession( ) in the AppLogic class saves data to the distributed state store *and* sets a cookie.
- The AppLogic should call saveSession( ) to save changes after updating session data with setSessionData( ) or after modifying the IValList returned by getSessionData( ).
- To improve performance, keep smaller amounts of information in the session.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
createSession( ) and saveSession( ) in the AppLogic class

setSessionData( )

Updates session with specified data.

Syntax
```
public int setSessionData(
   IValList pSessionData)
```

pSessionData. The IValList object containing the session data to set.

Usage
Use setSessionData( ) to write or update session data. Session data is stored in a distributed state layer in the iPlanet Application Server, making session data accessible to distributed server processes.

Tips
- The AppLogic should call saveSession( ) to save changes after updating session data with setSessionData( ).
- To improve performance, keep smaller amounts of information in the session.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
createSession( ) in the AppLogic class

ISessionIDGen Interface *(deprecated)*

The ISessionIDGen interface is deprecated and is provided for backward compatibility only. New applications should use the interface javax.servlet.http.HttpSession and the servlet programming model.

The ISessionIDGen interface represents a session ID generator. The session-related methods in the AppLogic class take an ISessionIDGen object as a parameter. By default, iPlanet Application Server uses the ISessionIDGen object to generate a session ID when an AppLogic creates a new session with the createSession( ) method in the AppLogic class.

The session ID—based on a 64-bit number—uniquely identifies a session between a user and an application. In a Web-based application, session IDs are passed between the Web browser and iPlanet Application Server to verify user sessions as users traverse the application. For non-browser clients, session IDs are tracked on the server.

Package

com.kivasoft

Related Topics

createSession( ) and getSession( ) in the AppLogic class (deprecated)

ISession2 Interface (deprecated)

IState2 Interface

The IState2 interface represents a node, or state object, in the State tree. A state tree is a hierarchical data storage mechanism. It is used primarily for storing application data that needs to be distributed across server processes and clusters. For example, the session data your application creates and maintains is stored in nodes of a state tree.

Use a state tree in your application if it needs to maintain and share data in a multi-server environment running load-balanced application components. A node has the following attributes:

- A name. Nodes on the same level of the state tree must have unique names, but not otherwise.
- Contents in the form of an IValList.
- A timeout value. You can specify if the content of the node automatically expires after a specified time. If you don’t specify a timeout value (timeout = 0), the content is saved until the node is deleted explicitly.
- Scope. You can specify if the node data is available at the local, cluster, or enterprise-wide level.

The IState2 interface defines methods for creating and deleting nodes, setting and retrieving node contents, and retrieving the attributes of a node.

To create a state tree, use the following methods:

- getStateTreeRoot( ) method in the AppLogic class to create the root node.
- createStateChild( ) in this interface to create the child nodes.

Package

com.kivasoft

Methods

Method	Description
createStateChild( )	Creates a child node under the node on which this method is called.
deleteStateChild( )	Deletes a child node.
getStateChild( )	Gets a specified child node.
getStateChildCount( )	Gets the count of children nodes.
getStateContents( )	Gets the contents of the node.
getStateFlags( )	Gets the flags assigned to the node when it was created.
getStateName( )	Returns the name of the node.
getStateTimeout( )	Returns the node's timeout value in seconds.
saveState( )	Saves updates to the node contents.
setStateContents( )	Sets node contents.

createStateChild( )

Creates a child node under the node on which this method is called.

Syntax

```
public IState2 createStateChild(
    String pName,
    int Timeout,
    int dwFlags)
```

pName. The name of the child node. If a child node with the given name already exists, this method returns an error.

Timeout. The unit of timeout is seconds. The meaning of timeout depends on the timeout flag specified in dwFlags. A value of 0 means the contents of the node is saved until deleted explicitly. You can assign a non-zero timeout value only to child nodes that are leaf nodes. Parent nodes can only have a timeout value of 0.

dwFlags. Specify one of the following flags, or 0 to use the default system settings:

- GXSTATE.GXSTATE_LOCAL to make the node visible to the local process only.
- GXSTATE.GXSTATE_CLUSTER to make the node visible to all application components within the cluster.
- GXSTATE.GXSTATE_DISTRIB to make the node visible to all application components in the enterprise environment.
- GXSTATE.GXSTATE_TIMEOUT_CREATE to specify that the contents of the node expires *n* seconds from the time the node was created.

The default scope is distributed and the default timeout is 60 seconds from the time the node was last accessed.

Usage

Use createStateChild( ) to add a child node to a state tree. The application component should already have created the root node of the tree with getStateTreeRoot( ) in the AppLogic class.

To create a new child node in a particular position of the tree, traverse the tree until you reach the node that will be the parent of the new child node. Then call createStateChild( ).

Rules

- The scope of a parent node must be the same as or greater than the scope of its child nodes. For example, if the scope of a child node is set to the cluster level, its parent node must be set to either the cluster or distributed level.
- Parent nodes can only have a timeout value of 0.

Tips

- To traverse the state tree to find the desired location in which to create a new child node, use getStateChild( ). Each successive call to getStateChild( ) descends one level in the tree.

Return Value
An IState2 object, or null for failure.

Example
The following code shows how to create a child node if it doesn't already exist:

```
IState2 tree = getStateTreeRoot(GXSTATE.GXSTATE_DISTRIB, "Grammy");


if (tree!=null)
{
   IState2 child = tree.getStateChild("Best Female Vocal");
   if (child == null)
   {
      child = tree.createStateChild("Best Female Vocal", 0,
        GXSTATE.GXSTATE_DISTRIB);
```

deleteStateChild( )
Deletes a child node from a state tree.

Syntax
```
public int deleteStateChild(
   String pName)
```

pName. The name of the child node to delete.

Usage
Use deleteStateChild( ) to delete a child node from a state tree when your application no longer needs it. A child node can be deleted only from its parent node. For example, if the state tree has three levels and you want to delete a node at the third level, traverse the tree until you find its parent node at the second level. Then call deleteStateChild( ) to delete a specific node

Rule
You can delete a parent node only after deleting its child nodes.

Tip
To traverse the state tree to find the parent node of the child node to delete, use getStateChild( ). Each successive call to getStateChild( ) descends one level in the tree.

Return Value
GXE.SUCCESS if the method succeeds.

getStateChild( )

Gets a specified child node.

Syntax
```
public IState2 getStateChild(
   String pName)
```

pName. The name of the child node to get.

Usage
Use getStateChild( ) to retrieve a node whose content you want to get or update. Your application component can also use getStateChild( ) to traverse a state tree to find the parent node of child nodes to add or delete.

Return Value
An IState2 object, or null for failure.

getStateChildCount( )

Gets the count of children nodes.

Syntax
```
public int getStateChildCount(
   int dwFlags)    // parameter is currently unused.
```

Usage
Use this method to return the number of children at any given state node.

Return Value
An integer representing the number of children.

getStateContents( )

Gets the contents of the node.

Syntax
```
public IValList getStateContents()
```

Usage

Use getStateContents( ) to retrieve the contents of the node, or to check if the node contains contents before setting values in the node. This method retrieves the contents that were last saved in the distributed store with saveState( ).

Tips

- To traverse the state tree to find a specific node, use getStateChild( ). Each successive call to getStateChild( ) descends one level in the tree.
- If you update the contents of a node with setStateContents( ) but do *not* save the contents in the distributed store with saveState( ), getStateContents( ) will not return the content set with setStateContents( ). It will return the contents that were last saved.

Return Value

An IValList object that contains the contents, or null for failure.

getStateFlags( )

Gets the flags assigned to the node when it was created.

Syntax

```
public int getStateFlags()
```

Usage

Use getStateFlags( ) to retrieve the flag that represents the node's scope, lifetime, and timeout criteria. This flag is specified when the state node is created. The following table describes the valid session flags, which are static variables in the GXSTATE class:

Flag	Description
GXSTATE_LOCAL	The node is visible to the local process only.
GXSTATE_CLUSTER	The node is visible to all application components within the cluster.
GXSTATE_DISTRIB	The node is visible to all application components in the enterprise environment.
GXSTATE_TIMEOUT_CREATE	The contents of the node expires *n* seconds from the time the node was created.

Return Value
One of the flags listed in the previous table.

getStateName( )

Returns the name of the node.

Syntax
```
public String getStateName()
```

Usage
Use getStateName( ) when the name of the node is unknown and is required for subsequent operations.

Return Value
A string value representing the name of the current node.

getStateTimeout( )

Returns the node's timeout value in seconds.

Syntax
```
public int getStateTimeout()
```

Usage
Use getStateTimeout( ) in conjunction with getStateFlags( ) to determine if and when the contents of the node expires. A timeout value of 0 means the node contents are saved until the node is deleted explicitly.

Return Value
Integer representing the timeout value in seconds.

saveState( )

Saves updates to the node contents.

Syntax
```
public int saveState(
    int dwFlags)
```

dwFlags. Specify 0 (zero). Internal use only.

Usage

Use saveState( ) after you set or change the contents of a node. This method flushes the node contents into the distributed store.

Tip

The getStateContents( ) method retrieves the contents that were last saved in the distributed store with saveState( ). Therefore, if you update the contents of a node with setStateContents( ), but do *not* call saveState( ), getStateContents( ) will not return the content set with setStateContents( ).

Return Value

GXE.SUCCESS if the method succeeds.

setStateContents( )

Sets node contents.

Syntax

```
public int setStateContents(
    IValList pContents)
```

pContents. IValList of values to set in the current node.

Usage

Use setStateContents( ) to update the contents of a node.

Tips

- To traverse the state tree to find the child node to update, use getStateChild( ). Each successive call to getStateChild( ) descends one level in the tree.
- Call saveState( ) after you set or change the contents of a node. This method flushes the node contents into the distributed store. If you call setStateContents( ) several times before calling saveState( ), only the value from the last setStateContents( ) call is saved.
- The getStateContents( ) method retrieves the contents that were last saved in the distributed store with saveState( ). Therefore, if you update the contents of a node with setStateContents( ), but do *not* call saveState( ), getStateContents( ) will not return the content set with setStateContents( ).

Return Value

GXE.SUCCESS if the method succeeds.

Example
The following code shows how to create a child node and set its contents:

IStreamBuffer Interface *(deprecated)*

The IStreamBuffer interface is deprecated and is provided for backward compatibility only. New applications developed according to the servlet-JSP programming model do not need the functionality provided by the IStreamBuffer interface.

The IStreamBuffer interface represents a buffer for capturing streamed output during template processing. Use a stream buffer if your AppLogic needs to manipulate the data before sending it to another AppLogic. For example, the AppLogic can collect the data in a stream buffer, then parse it or save it to a file.

To capture the data in a stream buffer, use the evalOutput( ) method in the AppLogic class and pass in an IStream object. To manipulate the data in the stream buffer, use the getStreamData( ) method in this interface.

To create an instance of the IStreamBuffer interface, use the GX.CreateStreamBuffer( ) method.

Package

com.kivasoft

Method

getStreamData( )	Returns an array of byte values from the stream buffer.

getStreamData( )

Returns an array of byte values from the stream buffer.

Syntax

```
public byte[] getStreamData(
   int flags)
```

flags. Specify 0 (zero).

Usage

Use getStreamData( ) to retrieve the contents of the stream buffer that was captured during streamed template processing. The AppLogic can then manipulate the data as needed.

Rule

Call getStreamData( ) after evalOutput( ) in the AppLogic class. The evalOutput( ) method captures output in the stream buffer if the AppLogic passes in an IStream object.

Return Value

Array of byte values, or null for failure.

Related Topics

evalOutput( ) in the AppLogic class (deprecated)

ITable Interface *(deprecated)*

ITable is deprecated and is provided for backward compatibility only. New applications should use the java.sql.DatabaseMetaData interface from the JDBC Core API.

The ITable interface represents the definition of a table that is part of a relational data source. ITable provides methods to perform the following types of operations:

- Add, update, and delete rows in the table.
- Obtain information about table attributes as they are defined in the database catalog. Table attributes include the table name, table columns, data connection, and so on. To obtain additional information about individual columns, use the methods in the IColumn Interface (deprecated).

The ITable interface is part of the Data Access Engine (DAE) service.

To create an instance of the ITable interface, use getTable( ) in the IDataConn Interface (deprecated) or getTable( ) in the IColumn Interface (deprecated).

Each call to getTable( ) returns a new ITable object rather than returning an existing table object.

Package

com.kivasoft

Methods

Method	Description
addRow( )	Inserts a new row in the table.
allocRow( )	Allocates a new, empty row buffer, replacing the previous row buffer if one exists.
deleteRow( )	Deletes one or more rows in the table.
enumColumnReset( )	Resets the column enumeration to the first column in the table.
enumColumns( )	Returns the definition of the next column in the table.
getColumn( )	Returns the definition of a column with the specified name.
getColumnByOrd( )	Returns the definition of the column in the specified ordinal position.
getColumnOrdinal( )	Returns the ordinal position of the column specified by name.
getDataConn( )	Returns the data connection object associated with the data source in which the table is defined.
getName( )	Returns the name of the table.
getNumColumns( )	Returns the number of columns in the table object.
setValueBinary( )	Specifies a BINARY value of a column in the row buffer.
setValueBinaryPiece( )	Specifies a LONG BINARY value of a column in the row buffer.
setValueDateString( )	Specifies the Date value of a column in the row buffer.
setValueDouble( )	Specifies the double value of a column in the row buffer.
setValueInt( )	Specifies the int value of a column in the row buffer.
setValueString( )	Specifies the String value of a column in the row buffer.
setValueText( )	Specifies a TEXT value of a column in the row buffer.
setValueTextPiece( )	Specifies a LONGTEXT value of a column in the row buffer.
updateRow( )	Modifies one or more rows in the table with the contents of the row buffer.

Example

```
// Add a new user to the User table
ITable tbl = conn.getTable("CTLuser");
tbl.allocRow();
```

```
tbl.setValueInt(tbl.getColumnOrdinal("UserID"),
nameParm.hashCode());
tbl.setValueString(tbl.getColumnOrdinal("AccessLevel"), aclStr);
tbl.setValueString(tbl.getColumnOrdinal("LoginName"), nameParm);
tbl.setValueString(tbl.getColumnOrdinal("Password"), passwordParm);
tbl.setValueDate(tbl.getColumnOrdinal("AddDate"), addDateParm);
tbl.setValueDouble(tbl.getColumnOrdinal("LoginTime"), TimeParm);
if(tbl.addRow(0, null) == 0)
   return result(valOut, "User added");
else
   return result(valOut, "Failed to add the user");
```

addRow( )

Inserts a new row in the table.

Syntax

```
public int addRow(
   int dwFlags,
   ITrans pTrans)
```

dwFlags. Specifies one of the following flags used to execute this insert operation:

- For synchronous operations, the default, specify zero or GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_SYNC.
- For asynchronous operations, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC.

pTrans. ITransobject that contains the transaction associated with this insert operation, or null.

Usage

Use addRow( ) to insert a new record into a table.

Rules

- Before adding a row, the AppLogic must first call allocRow( ) to create a row buffer.
- Next, the AppLogic must specify data values for the new row by calling any of the setValueXXX( ) methods, such as setValueString( ) or setValueBinary( ).

- The AppLogic must specify a value for any column defined as NOT NULL and without a DEFAULT value, such as keys.
- The AppLogic must be logged into the database with sufficient privileges to insert records in the target table.
- If the insert operation is part of a transaction, the AppLogic must first create an instance of the ITrans Interface (deprecated) using createTrans( ) in the AppLogic class (deprecated). The AppLogic must then call begin( ) before executing the statement and, after executing the statement, call commit( ) or rollback( ) as appropriate.

Tips

- To determine whether a column is defined as NOT NULL, use getNullsAllowed( ) in the IColumn Interface (deprecated).
- Alternatively, the AppLogic can insert records by passing a SQL INSERT statement using setSQL( ) in the IQuery Interface (deprecated). The statement must comply with ANSI 92 SQL syntax.

Return Value

An int value indicating success (zero) or failure (non-zero, such as an invalid transaction object).

Example

```
// Get a table

ITable table = conn.getTable("OBTransaction");

if (table == null)

{

   return handleOBSystemError("Could not access table
OBTransactionType.");

}


// Look up column ordinals for table

int transTypeCol = table.getColumnOrdinal("transType");

int postDateCol  = table.getColumnOrdinal("postDate");

int acctNumCol   = table.getColumnOrdinal("acctNum");

int amountCol    = table.getColumnOrdinal("amount");
```

```
// Create transaction object
ITrans transferTrans = createTrans();
if (transferTrans == null)
{
   return handleOBSystemError("Could not start transaction");
}
transferTrans.begin();
int rc;

// Allocate row in table for new entry
table.allocRow();
table.setValueString(acctNumCol, fromAcct);
table.setValueInt(transTypeCol, OBDBDefs.TRANSTYPE_WITHDRAWAL);
table.setValueDateString(postDateCol, transDateString);
table.setValueDouble(amountCol, amount.doubleValue() * -1.0);

// Add the row to the table
rc = table.addRow(0, transferTrans);
transferTrans.commit(0);
```

Related Topics
ITrans Interface (deprecated)

allocRow( )

Allocates a new, empty row buffer, replacing the previous row buffer if one exists.

Syntax
```
public int allocRow()
```

Usage
Use allocRow( ) to allocate a new row buffer before adding or updating records in a table. The row buffer is a virtual representation of a row in the target table, including all column definitions. The AppLogic writes data values to the row buffer first, then writes the contents of the row buffer to either a new record using addRow( ) or to one or more existing records using updateRow( ).

Rules

- The AppLogic must call allocRow( ) before specifying column values with a setValueXXX( ) method.
- The AppLogic must call allocRow( ) every time before calling addRow( ) or updateRow( ).

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Get a table
ITable table = conn.getTable("OBTransaction");
if (table == null)
{
   return handleOBSystemError("Could not access table
OBTransactionType.");
}

// Look up column ordinals for table
int transTypeCol = table.getColumnOrdinal("transType");
int postDateCol  = table.getColumnOrdinal("postDate");
int acctNumCol   = table.getColumnOrdinal("acctNum");
int amountCol    = table.getColumnOrdinal("amount");

// Create transaction object
ITrans transferTrans = createTrans();
if (transferTrans == null)
{
   return handleOBSystemError("Could not start transaction");
}
transferTrans.begin();
int rc;

// Allocate row in table for new entry
```

```
table.allocRow();
table.setValueString(acctNumCol, fromAcct);
table.setValueInt(transTypeCol, OBDBDefs.TRANSTYPE_WITHDRAWAL);
table.setValueDateString(postDateCol, transDateString);
table.setValueDouble(amountCol, amount.doubleValue() * -1.0);

// Add the row to the table
rc = table.addRow(0, transferTrans);
```

deleteRow( )

Deletes one or more rows in the table.

Syntax

```
public int deleteRow(
   int dwFlags,
   String szWhere,
   ITrans pTrans)
```

dwFlags. Specifies one of the following flags used to execute this delete operation:

- For synchronous operations, the default, specify zero or GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_SYNC.
- For asynchronous operations, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC.

szWhere. Selection criteria expression for one or more rows to delete. The syntax is the same as the SQL WHERE clause, only without the WHERE keyword. Use ANSI 92-compliant syntax. If an empty string is specified, all rows in the table are deleted.

pTrans. ITrans object that contains the transaction associated with this delete operation, or null.

Rules

- The AppLogic must be logged into the database with sufficient privileges to delete records in the target table.

- If the delete operation is part of a transaction, the AppLogic must first create an instance of the ITrans Interface (deprecated) using createTrans( ) in the AppLogic class (deprecated). The AppLogic must then call begin( ) before executing the statement and, after executing the statement, call commit( ) or rollback( ) as appropriate.

Tip

Alternatively, the AppLogic can delete records by passing a SQL DELETE statement using setSQL( ) in the IQuery Interface (deprecated), then executing the query. The statement must comply with ANSI 92 SQL syntax.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
ITable tbl=conn.getTable("CTLcatalog");

if(tbl==null)

   return result("Cannot find table: "+"CTLcatalog");

// Otherwise, delete the row with product specified in prodStr

tbl.deleteRow(0, "ProductID"+"="+prodStr, null);
```

Related Topics

ITrans Interface (deprecated)

enumColumnReset( )

Resets the column enumeration to the first column in the table.

Syntax

```
public int enumColumnReset()
```

Usage

Use enumColumnReset( ) before iterating through and retrieving columns in a table. enumColumnReset( ) ensures that column retrieval starts from the first column.

Thereafter, use enumColumns( ) to retrieve each column sequentially. Each enumColumns( ) call returns an IColumn object for the next column.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
String htmlString;
IColumn col;
ITable tbl = conn.getTable("Products");
htmlString += "<h2>Products Table</h2>";
tbl.enumColumnReset();
while ((col = tbl.enumColumns()) != null) {
   htmlString += "Column Name = ";
   htmlString += col.getName();
   htmlString += ", Column Type = ";
   htmlString += col.getType();
   htmlString += "<br>";
};
return result(htmlString)
```

enumColumns( )

Returns the definition of the next column in the table.

Syntax

```
public IColumn enumColumns()
```

Usage

Use enumColumns( ) when the column definition is unknown and required for subsequent operations. The AppLogic can use the returned IColumn object to determine characteristics of the column, such as its name, data type, size, whether nulls are allowed, and so on.

Before iterating through columns, the client code should call enumColumnReset( ) to ensure that enumColumns( ) starts with the first column in the table. Each subsequent enumColumns( ) call moves to the next sequential column in the table and retrieves its column definition in an IColumn object.

Tips

- The columns might not be returned in the order in which they are defined in the database catalog.
- Test for null to determine when the last column has been retrieved.

Return Value

IColumn object containing the next column of data, or null for failure (such as no more columns in the table).

Example

```
String htmlString;

IColumn col;

ITable tbl = conn.getTable("Products");

htmlString += "<h2>Products Table</h2>";

tbl.enumColumnReset();

while ((col = tbl.enumColumns()) != null) {
   htmlString += "Column Name = ";
   htmlString += col.getName();
   htmlString += ", Column Type = ";
   htmlString += col.getType();
   htmlString += "<br>";
};

return result(htmlString)
```

Related Topics

IColumn Interface (deprecated)

getColumn( )

Returns the definition of a column with the specified name.

Syntax

```
public IColumn getColumn(
   String szColumn)
```

szColumn. Name of the column to retrieve.

Usage

Use getColumn( ) when the column definition is unknown but its name is known. The AppLogic can use the IColumn object to determine other characteristics about the column, such as its data type, size, whether nulls are allowed, and so on.

Rule
The specified column name must exist in the table.

Return Value
IColumn object for the specified column, or null for failure (such as an invalid column name).

Example
```
ITable tbl = conn.getTable("Products");

// Obtain column definitions

IColumn colProd = tbl.getColumn("ProductId")

IColumn colCat = tbl.getColumn("CategoryId")

IColumn colPrdName = tbl.getColumn("ProductName")

IColumn colPrice = tbl.getColumn("Price")

. . . manipulate columns using IColumn methods . . .
```

Related Topics
IColumn Interface (deprecated)

getColumnByOrd( )
Returns the definition of the column in the specified ordinal position.

Syntax
```
public IColumn getColumnByOrd(
   int Ordinal)
```

Ordinal. Ordinal number (position) of the column in the table. The first column is 1, the second column is 2, and so on.

Usage
Use getColumnByOrd( ) when the column definition is unknown but its position in the table is known, such as when iterating through columns in the table. The AppLogic can use the IColumn object to determine other characteristics about the column, such as its name, data type, size, whether nulls are allowed, and so on.

Rule
The specified column number must exist in the table.

Tips
- Column positions in a table may change between different table objects.

- Columns are not guaranteed to be in the same order in which the database lists them.
- To iterate through columns in a table using getColumnByOrd( ), call getNumColumns( ) to determine the maximum number of columns in the table, then proceed sequentially through each column using getColumnByOrd( ), beginning with column 1, through the last column.
- Alternatively, call enumColumnReset( ) to start with the first column in the table, then call enumColumns( ) repeatedly through the last column.

Return Value

IColumn object for the specified column, or null for failure (such as an invalid column number).

Example

```
// Obtain information about all columns in the table
ITable tbl = conn.getTable("Products");
int i = 1;
int maxCols = tbl.getNumColumns();
for (i <= maxCols; i++) {
   IColumn col = tbl.getColumnByOrd(i);
   . . . manipulate column using IColumn methods . . .
}
```

Related Topics

IColumn Interface (deprecated)

getColumnOrdinal( )

Returns the ordinal position of the column specified by name.

Syntax

```
public int getColumnOrdinal(
   String szColumn)
```

szColumn. Name of the column.

Usage

Use getColumnOrdinal( ) when the ordinal position of a column is unknown and is required for subsequent operations. For example, the ordinal position of a column is a required parameter value for the setValue**( ) methods, such as setValueString( ) and setValueInt( ).

Rule

The specified column name must exist in the table.

Return Value

An int value representing the ordinal position of the specified column, or zero for failure (such as an invalid column name). The first column is 1, the second column is 2, and so on.

Example

```
// Get a table

ITable table = conn.getTable("OBTransaction");

if (table == null)

{

   return handleOBSystemError("Could not access table
OBTransactionType.");

}


// Look up column ordinals for table

int transTypeCol = table.getColumnOrdinal("transType");

int postDateCol  = table.getColumnOrdinal("postDate");

int acctNumCol   = table.getColumnOrdinal("acctNum");

int amountCol    = table.getColumnOrdinal("amount");
```

Related Topics

IColumn Interface (deprecated)

getDataConn( )

Returns the data connection object associated with the data source in which the table is defined.

Syntax

```
public IDataConn getDataConn()
```

Usage
Use getDataConn( ) when the data connection associated with the table is unknown and is required for subsequent operations.

Tip
The IDataConn object that getDataConn( ) returns may not be equal (==) to the IDataConn object that createDataConn( ), in the AppLogic class (deprecated), returned.

Return Value
IDataConn object for the table, or null for failure.

Related Topics
IDataConn Interface (deprecated)

getName( )
Returns the name of the table.

Syntax
```
public String getName()
```

Usage
Use getName( ) when the name of the table is unknown and is required for subsequent operations.

Return Value
Name of the table represented by the ITable object, or null for failure (such as a memory allocation error).

getNumColumns( )
Returns the number of columns in the table object.

Syntax
```
public int getNumColumns()
```

Usage

Use getNumColumns( ) when the number of columns defined in the table is unknown and is required for subsequent operations. When iterating through columns in a table, the AppLogic can use this information to specify the maximum number of iterations.

Return Value

An int value representing the number of columns in the table object, or zero for failure (such as a table with no columns defined).

Example

```
// Obtain information about all columns in the table

ITable tbl = conn.getTable("Products");

int i = 1;

int maxCols = tbl.getNumColumns();

for (i <= maxCols; i++) {

   IColumn col = tbl.getColumnByOrd(i);

   . . . manipulate column using IColumn methods . . .

}
```

setValueBinary( )

Specifies a BINARY value of a column in the row buffer.

Syntax

```
public int setValueBinary(
   int Ordinal,
   byte[] pValue,
   int nOffset,
   int nLength)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

pValue. A byte array expression to assign to the column.

nOffset. Number of bytes to skip from the beginning of the byte array. This value specifies the starting point within the array.

nLength. Number of bytes to set for the byte array.

Usage

Use setValueBinary( ) for BINARY data of which the total size is equal to or smaller than 64K.

Rules

- The AppLogic must call allocRow( ) before attempting to write to the row buffer.
- The data type of the column must be BINARY or VARBINARY, or database equivalent.

Tip

Use setValueBinaryPiece( ) for LONGBINARY, LONGVARBINARY, or equivalent type values.

Return Value

GXE.SUCCESS if the method succeeds.

setValueBinaryPiece( )

Specifies a LONGBINARY value of a column in the row buffer.

Syntax

```
public int setValueBinaryPiece(
   int Ordinal,
   byte[] pValue,
   int nOffset,
   int nLength)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

pValue. A byte array expression to assign to the column.

nOffset. Number of bytes to skip from the beginning of the byte array. This value specifies the starting point within the array.

nLength. Number of bytes to set for the byte array.

Usage

Use setValueBinaryPiece( ) to specify LONGBINARY data. LONGBINARY data must be added in 64K increments, therefore, you must use setValueBinaryPiece( ) several times to add the data.

Rules

- The AppLogic must call allocRow( ) before attempting to write to the row buffer.
- The data type of the column must be LONGBINARY, LONGVARBINARY, or database equivalent.
- Must be called after allocRow( ) but before addRow( ) or updateRow( ).

Tip

Use setValueBinary( ) for BINARY, VARBINARY, or equivalent type values.

Return Value

GXE.SUCCESS if the method succeeds.

setValueDateString( )

Specifies the Date value of a column in the row buffer.

Syntax

```
public int setValueDateString(
   int Ordinal,
   String pValue)
```

Ordinal. Ordinal number (position) of the target column in the table. The first column is 1, the second column is 2, and so on.

pValue. A date expression to assign to the column. Use one of the following formats:

- "Fri Oct 10 14:35:59.999 PDT 1997"

  The subseconds (,999 in the example) and time zone (PDT in the example) are optional.

- "1997-10-01 14:35:59.999"

  The time is optional.

Rule

The AppLogic must call allocRow( ) before attempting to write to the row buffer.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Get a table

ITable table = conn.getTable("OBTransaction");

if (table == null)

{

   return handleOBSystemError("Could not access table
OBTransactionType.");

}

// Look up a column ordinal

int postDateCol = table.getColumnOrdinal("postDate");


// Allocate a new row and set a datestring value

table.allocRow();

table.setValueDateString(postDateCol, transDateString);
```

setValueDouble( )

Specifies the double value of a column in the row buffer.

Syntax

```
public int setValueDouble(
   int Ordinal,
   double nValue)
```

Ordinal. Ordinal number (position) of the target column in the table. The first column is 1, the second column is 2, and so on.

nValue. A double expression to assign to the column.

Rule

The AppLogic must call allocRow( ) before attempting to write to the row buffer.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Get a table

ITable table = conn.getTable("OBTransaction");

if (table == null)
```

```
{
   return handleOBSystemError("Could not access table
   OBTransactionType.");
}

// Look up a column ordinal
int amountCol  = table.getColumnOrdinal("amount");

// Allocate a new row and set a double value
table.allocRow();
table.setValueDouble(amountCol, amount.doubleValue() * -1.0);
```

setValueInt( )

Specifies the int value of a column in the row buffer.

Syntax

```
public abstract int setValueInt(
   int Ordinal,
   int nValue)
```

Ordinal. Ordinal number (position) of the target column in the table. The first column is 1, the second column is 2, and so on.

nValue. An int expression to assign to the column.

Rule

The AppLogic must call allocRow( ) before attempting to write to the row buffer.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Get a table
ITable table = conn.getTable("OBTransaction");
if (table == null)
{
   return handleOBSystemError("Could not access table
```

```
   OBTransactionType.");
}

// Look up a column ordinal
int transTypeCol = table.getColumnOrdinal("transType");

// Allocate a new row and set an Int value
table.allocRow();
table.setValueInt(transTypeCol, OBDBDefs.TRANSTYPE_WITHDRAWAL);
```

setValueString( )

Specifies the String value of a column in the row buffer.

Syntax

```
public int setValueString(
   int Ordinal,
   String pValue)
```

Ordinal. Ordinal number (position) of the target column in the table. The first column is 1, the second column is 2, and so on.

pValue. A String expression to assign to the column.

Rule

The AppLogic must call allocRow( ) before attempting to write to the row buffer.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Get a table
ITable table = conn.getTable("OBTransaction");
if (table == null)
{
   return handleOBSystemError("Could not access table
   OBTransactionType.");
}
```

```
// Look up a column ordinal
int acctNumCol   = table.getColumnOrdinal("acctNum");


// Allocate a new row and set a string value
table.allocRow();
table.setValueString(acctNumCol, fromAcct);
```

setValueText( )

Specifies a TEXT value of a column in the row buffer.

Syntax

```
public int setValueText(
    int Ordinal,
    String pValue,
    int nOffset,
    int nLength)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

pValue. A string expression to assign to the column.

nOffset. Number of characters to skip from the beginning of the string.

nLength. Number of characters to set.

Usage

Use setValueText( ) for TEXT data, or database equivalent, of which the total size is equal to or smaller than 64K.

Rules

- The AppLogic must call allocRow( ) before attempting to write to the row buffer.
- The data type of the column must be TEXT or database equivalent.

Tip

Use setValueTextPiece( ) for LONGTEXT or equivalent type values.

Return Value

GXE.SUCCESS if the method succeeds.

setValueTextPiece( )

Specifies a LONG TEXT value of a column in the row buffer.

Syntax

```
public int setValueText(
    int Ordinal,
    String pValue,
    int nOffset,
    int nLength)
```

Ordinal. Ordinal number (position) of the column in the table definition. The first column is 1, the second column is 2, and so on.

pValue. A string expression to assign to the column.

nOffset. Number of characters to skip from the beginning of the string.

nLength. Number of characters to set.

Usage

Use setValueTextPiece( ) for LONGTEXT data. LONGTEXT values must be added in 64K increments, therefore, you must call setValueTextPiece( ) repeatedly to add the data.

Rules

- The AppLogic must call allocRow( ) before attempting to write to the row buffer.
- The data type of the column must be LONGTEXT or database equivalent.

Tip

Use setValueText( ) for TEXT or equivalent type values.

Return Value

GXE.SUCCESS if the method succeeds.

updateRow( )

Modifies one or more rows in the table with the contents of the row buffer.

Syntax

```
public int updateRow(
    int dwFlags,
    String szWhere,
    ITrans pTrans)
```

dwFlags. Specifies one of the following flags used to execute this update operation:

- For synchronous operations, the default, specify zero or GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_SYNC.
- For asynchronous operations, specify GX_DA_EXECUTEQUERY_FLAGS.GX_DA_EXEC_ASYNC.

szWhere. Selection criteria expression for one or more rows to update. The syntax is the same as the SQL WHERE clause, only without the WHERE keyword. Use ANSI 92-compliant syntax. If an empty string is specified, all rows in the table are updated.

pTrans. ITrans object that contains the transaction associated with this update operation, or null.

Rules

- Before modifying a row, the AppLogic must first call allocRow( ) to create the row buffer.
- Next, the AppLogic must specify data values for the new row by calling any of the following methods: setValueDateString( ), setValueDouble( ), setValueInt( ) , or setValueString( ).
- For tables defined with one or more UNIQUE keys, the AppLogic can perform a single-record update but not a multiple-record update.
- The AppLogic must specify a value for any column defined as NOT NULL and without a DEFAULT value, such as keys.
- The AppLogic must be logged into the database with sufficient privileges to update records in the target table.
- If the update operation is part of a transaction, the AppLogic must first create an instance of the ITrans Interface (deprecated) using createTrans( ) in the AppLogic class (deprecated). The AppLogic must then call begin( ) before executing the statement and, after executing the statement, call commit( ) or rollback( ) as appropriate.

Tips

- The updateRow( ) method overwrites all columns in the target record(s) with the contents of the row buffer. Therefore, retrieve the row first using a query, assign the column values to the row buffer, then change only the column(s) you want to update.
- To determine whether a column is defined as NOT NULL, use getNullsAllowed( ) in the IColumn Interface (deprecated).
- Alternatively, the AppLogic can update records by passing a SQL INSERT statement using setSQL( ) in the IQuery Interface (deprecated). The statement must comply with ANSI 92 SQL syntax.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Obtain the definition for the Catalog table

ITable tbl=conn.getTable("CTLcatalog");

if(tbl==null)

   return result("Cannot find table: "+"CTLcatalog");

   tbl.allocRow();

   int cProductId=tbl.getColumnOrdinal("productId");

   int cCategory=tbl.getColumnOrdinal("category");

   int cName=tbl.getColumnOrdinal("name");

   int cPrice=tbl.getColumnOrdinal("price");

   int cDesc=tbl.getColumnOrdinal("desc");

   int cImage1=tbl.getColumnOrdinal("image1");

   int cTemplate=tbl.getColumnOrdinal("template");

   int cInventory=tbl.getColumnOrdinal("inventory");

   int cReorder=tbl.getColumnOrdinal("reorder");


if((cProductId*cCategory*cName*cPrice*cDesc*cImage1*cTemplate*cReor
der)

   ==0)

   return result("Cannot map columns on table:

   "+"Catalog");
```

```
// Set up the table columns
tbl.setValueInt(cCategory, category);
tbl.setValueString(cName, name);
tbl.setValueDouble(cPrice, price);
tbl.setValueString(cDesc, desc);
tbl.setValueString(cImage1, image1);
tbl.setValueString(cTemplate, template);
tbl.setValueInt(cInventory, inventory);
tbl.setValueInt(cReorder, reorder);
if(option==ADD)
  {
    valIn.setValString("productId", prodStr);
    tbl.setValueInt(cProductId, productId);
    tbl.addRow(0, null);
   }
else
  tbl.updateRow(0, "productId"+"="+prodStr, null);
```

Related Topics

ITrans Interface (deprecated)

ITemplateData Interface (*deprecated)*

ITemplateData is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model, where similar functionality is provided through interfaces such as java.sql.ResultSet and javax.sql.RowSet.

For information about replacing ITemplateData functionality in existing applications, see the *Migration Guide*.

The ITemplateData interface represents a hierarchical source of data used for HTML template processing. ITemplateData provides methods for iterating through rows in a set of memory-based hierarchical data and retrieving column values.

To create an ITemplateData object, an AppLogic calls GX.CreateTemplateDataBasic( ). The AppLogic populates the ITemplateData object with rows of hierarchical data, then passes this ITemplateDataBasicobject as the data parameter in evalTemplate( ) or evalOutput( ) in the AppLogic class (deprecated). The Template Engine then draws upon the hierarchical data during template processing using methods in the ITemplateData interface.

The Template Engine normally processes the hierarchical template data internally. To provide application-specific special processing and hook into the template generation process, the AppLogic can subclass the TemplateDataBasic class (deprecated) and override the ITemplateData member methods.

Package

com.kivasoft

Methods

Method	Description
getValue( )	The Template Engine calls this method to dynamically retrieve the value of the specified field from the current row in the hierarchical template data.
isEmpty( )	The Template Engine calls this method to determine whether the specified group in the hierarchical result set is empty (contains no rows).
moveNext( )	The Template Engine calls this method to retrieve the next row of the specified group in the hierarchical template data object.
setHint( )	Placeholder method for future functionality.

Example

```
// Determine whether AppLogic module is listed in project
TemplateDataBasic tdb;
tdb = GetProjectData();
if (tdb == null) {
   return false;
}
if (!tdb.isEmpty("MODULE")) {
   loopChild:
```

```
   do {
      String name;
      name = tdb.getValue("MODULENAME");
      if ((name != null) &&
         (name.compareTo(moduleName) == 0)) {
         GenerateReport("The AppLogic already exists.<br>"
          +"Please choose another name.", null, null);
            return false;
      }
   }
   while (tdb.moveNext("MODULE") == 0);
}
```

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

TemplateDataBasic class (deprecated)

getValue( )

The Template Engine calls this method to dynamically retrieve the value of the specified field from the current row in the hierarchical template data.

Syntax

```
public IBuffer getValue(
   String szExpr)
```

szExpr. Name of a field in the template data object.

Usage

The Template Engine calls getValue( ) to retrieve values from the hierarchical template data object for subsequent processing.

Rule

The specified field name must exist in the template data object.

Tips

- When processing result sets, first call isEmpty( ) to determine whether rows were returned. Next, for each row in the result set, call getValue( ) to retrieve field values, then call moveNext( ) to move to the next row in the result set, until the end of the result set is reached.
- Use methods in the IBuffer interface to manipulate the returned memory block.
- Use Util.toString( ) to convert the contents of the returned memory buffer into a String.

Return Value

An IBuffer object that contains the value of the specified field name, or null for failure (such as invalid field name).

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

TemplateDataBasic class (deprecated)

isEmpty( )

The Template Engine calls this method to determine whether the specified group in the hierarchical result set is empty (contains no rows).

Syntax

```
public boolean isEmpty(
   String group)
```

group. Name of a group in the hierarchical result set.

Usage

The Template Engine calls isEmpty( ) to test whether the specified group in the ITemplateData object contains any rows of data before processing individual fields using getValue( ).

Rule

The specified group name must exist in the hierarchical data set.

Tip

When processing result sets, first call isEmpty( ) to determine whether rows were returned. Next, for each row in the result set, call getValue( ) to retrieve field values, then call moveNext( ) to move to the next row in the result set, until the end of the result set is reached.

Return Value
True if the specified group is empty, and false if it contains rows of data or if the specified group name is invalid.

Related Topics
evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

TemplateDataBasic class (deprecated)

moveNext( )

The Template Engine calls this method to retrieve the next row of the specified group in the hierarchical template data object.

Syntax
```
public int moveNext(
   String group)
```

group. Name of a group to process in the hierarchical data of the template data object.

Usage
The Template Engine calls moveNext( ) when iterating through rows in the template data object to retrieve the contents of the next sequential hierarchical row of data.

Rule
The specified group name must exist in the hierarchical data set.

Tip
When processing result sets, first call isEmpty( ) to determine whether rows were returned. Next, for each row in the result set, call getValue( ) to retrieve field values, then call moveNext( ) to move to the next row in the result set, until the end of the result set is reached.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

TemplateDataBasic class (deprecated)

setHint( )

The setHint( ) method is a placeholder for future functionality. Currently, it is implemented to return 0. If you create a custom template data class that implements the ITemplateData interface, implement setHint( ) to return 0.

Syntax

```
public int setHint(
    String group,
    int flags,
    int max,
    IValList pVal)
```

ITemplateMap Interface *(deprecated)*

ITemplateMap is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model.

For information about replacing ITemplateMap functionality in existing applications, see the *Migration Guide*.

The ITemplateMap interface represents a mapping between a template field specification and dynamic data used for HTML template processing. ITemplateMap provides the get( ) method for resolving the `id` attribute in a GX markup tag. Each `id` attribute contains a field name that can be mapped.

To create a field map, an AppLogic calls GX.CreateTemplateMapBasic( ). The AppLogic then populates the field map using put( ), in the TemplateMapBasic class (deprecated), for each field mapping, then passes this ITemplateMap object as the map parameter in evalTemplate( ) or evalOutput( ) in the AppLogic class (deprecated). When the Template Engine encounters a GX markup tag with the `id` attribute while processing the template, it calls get( ) in the ITemplateMap Interface (deprecated) to resolve the name. The get( ) method, in turn, calls getString( ) in the TemplateMapBasic class (deprecated) to actually resolve the name.

To provide application-specific special processing, an AppLogic can subclass the TemplateMapBasic class (deprecated) and override the getString( ) method to hook into the Template Engine generation process. For example, the AppLogic can intercept and filter data from a database before the Template Engine processes it.

Package

com.kivasoft

Method

get( )	Resolves the id attribute specified in a GX markup tag in the template being processed by the Template Engine. This method is called by the Template Engine.

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

TemplateMapBasic class (deprecated)

ITemplateData Interface (deprecated)

TemplateDataBasic class (deprecated)

get( )

Resolves the `id` attribute specified in a GX markup tag in the template being processed by the Template Engine. This method is called by the Template Engine.

Syntax

```
public IBuffer get(
   String szExpr,
   IObject pData,
   IObject pMark)
```

szExpr. In the current GX markup tag in the HTML template being processed, the name of the field, or placeholder, assigned to the id attribute. Must be an identical match (case-sensitive).

pData. Specify null. Internal use only.

pMark. Specify null. Internal use only.

Usage

GX markup tags are used in an HTML template to identify where dynamic data appears in the output report. In the GX markup tags, the `id` attribute specifies any of the following items: the name of a flat query within a hierarchical query, a field in the hierarchical result set or TemplateDataBasic object, or an HTML template. The type of item specified in the `id` attribute depends on the `type` attribute that is specified in the same GX markup tag.

The Template Engine calls get( ) to resolve the`id` attribute specified in a GX markup tag in the template being processed by the Template Engine. The get( ) method, in turn, calls getString( ) in the TemplateMapBasic class (deprecated) to actually resolve the name. To provide application-specific special processing, an AppLogic can subclass the TemplateMapBasic class (deprecated) and override getString( ) to manipulate the Template Engine generation process. For example, an AppLogic can intercept and filter data from a database before the Template Engine processes it.

Return Value

An IBuffer object that contains the value of the id mapping, or null for failure.

Related Topics

evalTemplate( ) and evalOutput( ) in the AppLogic class (deprecated)

getString( ) in theTemplateMapBasic class (deprecated)

ITemplateData Interface (deprecated)

TemplateDataBasic class (deprecated)

ITile Interface *(deprecated)*

ITile is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model.

For information about replacing ITile functionality in existing applications, see the *Migration Guide*.

The ITile interface represents a tile, which is a record set that contains multiple records. A tile can also contain nested tiles. Organized like a hierarchical result set, a tile is returned by the GX.ProcessOutput( ) method.

AppLogics use ITile together with GX.ProcessOutput( ) when working with non-HTML results returned by another AppLogic. The following are the general steps for getting the tile:

1. A client AppLogic calls an AppLogic with newRequest( ).
2. Through newRequest( ), the client passes input and output IValLists to the called AppLogic. If the client is an AppLogic, it specifies the value "ocl" for the gx_client_type key in the input IValList.
3. The called AppLogic processes the request and sends back results using its output IValList or by calling evalOutput( ).

4. The client calls the GX.ProcessOutput( ) method to process the results into an ITile object.
5. Using methods in the ITile interface, the client traverses the tile and retrieves values to populate user interface controls, such as text boxes or list boxes, on a form.

The tile corresponds to the structure specified by the `tile` and `cell` tags in the template file that the called AppLogic used when it called evalOutput( ). The `tile` tag determines the tile or record set, and the `cell` tag, the values in each record.

Package

com.kivasoft

Methods

Method	Description
getTileChild( )	Returns the specified child tile.
getTileValue( )	Returns the value of a specified field in a record.
moveTileNextRecord( )	Moves to the next record in the tile.
moveTileToRecord( )	Moves to a specific record in the tile.

Example

The following example shows a template file used by a called AppLogic when generating output, and a section of a program that uses GX.ProcessOutput( ) and ITile methods to process the output:

GXML template file:

```
<gx type=tile id="PRODUCTS" max=100>
<gx type=cell id="PRODUCTS.Category"></gx>
<gx type=cell id="PRODUCTS.ProdName"></gx>
</gx>
<gx type=tile id="CATEGORIES" max=100>
<gx type=cell id="CATEGORIES.CategoryId"></gx>
</gx>
```

Code snippet:

```
// Call this AppLogic
hr = Conn.newRequest(guid, vIn, vOut, 0);
if (hr==GXE.SUCCESS)
{
   // Get the root tile from the output vallist
   ITile roottile = GX.ProcessOutput(null, 0, vOut);
   if (roottile != null)
   {
      String sval;
      ITile ptile;

      //Iterate over all products and print their names
      ptile = roottile.getTileChild("PRODUCTS");
      hr = GXE.SUCCESS;
      System.out.println("Products:");
      while (ptile != null && hr != GXE.FAIL)
      {
         sval = ptile.getTileValue("PRODUCTS.ProdName");
         System.out.println("   " + sval);

         hr = ptile.moveTileNextRecord();
      }

      //Iterate over all categories and print their ids
      ptile = roottile.getTileChild("CATEGORIES");
      hr = GXE.SUCCESS;
      System.out.println("Categories:");
      while (ptile != null && hr != GXE.FAIL)
      {
         sval = ptile.getTileValue("CATEGORIES.CategoryId");
         System.out.println("   " + sval);
```

```
            hr = ptile.moveTileNextRecord();
         }
      }
   }
```

Related Topics

newRequest( ) and evalOutput( ) in the AppLogic class (deprecated)

GX.ProcessOutput( ),
IValList Interface (deprecated)

getTileChild( )

Returns the specified tile.

Syntax

```
public ITile getTileChild(
   String name)
```

name. The name of the child tile in the tile. This name must match a name assigned to the `id` attribute of type `tile` in the template file.

Usage

Use getTileChild( ) to retrieve a tile from which to get records and record values. Use it in conjunction with moveTileNextRecord( ) and getTileValue( ) to traverse the tile and retrieve record values. The client can call these methods in a loop until all values in a tile have been retrieved.

Return Value

An ITile object, or null for failure.

Related Topics

getTileValue( ),
moveTileNextRecord( )

getTileValue( )

Returns the value of a specified field in a record.

Syntax

```
public String getTileValue(
   String name)
```

name. The name of the field in the current record. This name must match a name assigned to the `id` attribute of type `cell` in the template file.

Usage

Use getTileValue( ) to retrieve values in a record. Use it in conjunction with getTileChild( ) and moveTileNextRecord( ) to traverse the tile and retrieve each value. The client can call these methods in a loop until all values in a tile have been retrieved.

Return Value

The record value as a string.

Related Topics

getTileChild( ),
moveTileNextRecord( )

moveTileNextRecord( )

Moves to the next record in the tile.

Syntax

```
public int moveTileNextRecord()
```

Usage

Use moveTileNextRecord( )to go to the next record in a tile after retrieving values in the current record. Use the method in conjunction with getTileChild( ) and getTileValue( ) to traverse the tile and retrieve each value. The client can call these methods in a loop until all values in a tile have been retrieved.

Return Value

GXE.SUCCESS if the method succeeds.

Related Topics

getTileChild( ),
getTileValue( )

moveTileToRecord( )

Moves to a specific record in the tile.

Syntax

```
public int moveTileNextRecord(
    int ord)
```

ord. The position of the record in the tile. The first record in a tile is 1, the second is 2, and so on.

Usage

Use moveTileToRecord( ) when iterating through the tile multiple times. For example, after iterating through all the records, the AppLogic can return to the first record in preparation for the next iteration. If the tile contains many records, the AppLogic can also use moveTileToRecord( ) to display only several records at a time.

Return Value

GXE.SUCCESS if the method succeeds.

ITrans Interface *(deprecated)*

ITrans is deprecated and is provided for backward compatibility only. In new applications, either use XA transactions from the Java Transaction API (JTA), or set the appropriate autocommit mode on java.sql.Connection, an interface of the JDBC Core API.

The ITransinterface represents a transaction object used for subsequent transaction processing operations. ITrans provides operations for beginning, committing, and rolling back transactions.

After instantiating a transaction object, the AppLogic calls begin( ) to start the transaction. Next, the AppLogic performs any query, insert, update, or delete operations, passing the transaction object to the respective method in the ITable Interface (deprecated). Finally, the AppLogic closes the transaction by calling either commit( ) to save all changes or rollback( ) to cancel them. Closing a transaction terminates the transaction object and releases system resources.

The calls that make up a transaction can be in any part of the code; they need not be consecutive. The commands in a transaction are united by the fact that they all have the same transaction object as a parameter.

An application can process several transactions simultaneously. Each transaction works with a different database connection object. Within a single transaction, however, all the commands must access a single database through a single connection object.

To create an instance of the ITrans interface, use createTrans( ) in the AppLogic class (deprecated), as shown in the following example:.

```
ITrans trx = createTrans();
```

Package

com.kivasoft

Methods

Method	Description
begin( )	Starts the transaction.
commit( )	Commits the transaction, saving any changes.
rollback( )	Rolls back the transaction, abandoning any changes.

Example

```
// Create and begin a transaction
ITrans trx=createTrans();
trx.begin();
// 1) Process the credit card
if(!processCreditCard(cusId, card, number, expirationDate, trx)) {
   trx.rollback();
   return result("Could not process the credit card information"); }
// 2) Process the invoice record
InfoHolder info=new InfoHolder();
if(!makeInvoiceRecord(cusId, number, trx, info)) {
   trx.rollback();
   return result("Could not create the invoice record"); };
// 3) Process products on the invoice
if(!makeInvoiceEntries(info.invoiceId, trx)) {
   trx.rollback();
   return result("Could not create the invoice product records"); };
```

```
// 4) Process optional shipping information

if(shippingInfo && !makeShippingRecord(info.invoiceId, trx, addr1,
addr2, city, state, zip)) {

   trx.rollback();

   return result("Could not create the shipping information
record"); };

// 5) Process the inventory for each purchased product

if(!reduceProductInventory(trx)) {

   // Problem occurred - abandon everything

   trx.rollback();

return 0;

}

// No problem occurred - save everything

trx.commit(0);
```

Related Topics

createTrans( ) in the AppLogic class (deprecated)

addRow( ), updateRow( ), and deleteRow( ) in the ITable Interface (deprecated)

begin( )

Starts the transaction.

Syntax

```
public int begin()
```

Usage

Use begin( ) to start a transaction before performing any operations in the transaction. Subsequent operations belong to the current transaction until either commit( ) or rollback( ) is called.

Rules

- AppLogic must start the transaction explicitly using begin( ) before performing any query, insert, update, or delete operations associated with the transaction.

- AppLogic must complete the transaction explicitly by calling commit( ) to save any changes to tables or rollback( ) to abandon them. If a database error occurs before either are called, the database server will roll back the transaction automatically.

Tip

Use transactions judiciously to avoid locking conflicts. For example, avoid deadlocks by not using different open transactions on the same table.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Create a transaction
ITrans updCustTrans = createTrans();


if (updCustTrans == null)
{
   return handleOBSystemError("Could not create transaction");
}


// Begin the transaction
updCustTrans.begin();
```

Related Topics

createTrans( ) in the AppLogic class (deprecated)

addRow( ), updateRow( ), and deleteRow( ) in the ITable Interface (deprecated)

commit( )

Commits the transaction, saving any changes.

Syntax

```
public IObject commit(
   int dwFlags)
```

dwFlags. Specify 0.

Usage

Use commit( ) to commit a transaction and write unsaved changes to disk. commit( ) saves the changes, terminates the transaction object, and releases system resources.

Rules

- The AppLogic must start the transaction explicitly by calling begin( ) before any changes associated with the transaction can be committed.
- The AppLogic must complete the transaction explicitly by calling commit( ) to save any changes to tables or rollback( ) to abandon them.
- The AppLogic cannot reuse an ITrans object that has been committed. It must create a new one using createTrans( ) in the AppLogic class (deprecated).

Tips

- If an error occurs before the commit operation succeeds, the database server usually rolls back the transaction automatically.
- The target database server may take time to process a commit request.

Return Value

IObject object (for internal use only), or null for failure.

Example

```
// Update Customer record
rs = updCustPQuery.execute(0, custValList, updCustTrans, null);


if (rs == null)
{
   // Rollback transaction if the update operation failed
   updCustTrans.rollback();
   return handleOBSystemError("Could not update Customer.");
        }
// Commit the transaction if the update operation succeeded
updCustTrans.commit(0);


valIn.setValString("OUTPUTMESSAGE", "Successfully updated customer
record.");
```

Related Topics

createTrans( ) in the AppLogic class (deprecated)

addRow( ), updateRow( ), and deleteRow( ) in the ITable Interface (deprecated)

rollback( )

Rolls back the transaction, abandoning any changes.

Syntax

```
public int rollback()
```

Usage

Many database servers buffer changes made during a transaction, then update the affected tables only after the commit request is received.

Rolling back a transaction terminates the transaction object and releases system resources.

Rules

- The AppLogic must start the transaction explicitly by calling begin( ) before any changes associated with the transaction can be rolled back.
- The AppLogic must complete the transaction explicitly by calling commit( ) to save any changes to tables or rollback( ) to abandon them.
- The AppLogic cannot reuse an ITrans object that has been rolled back. It must create a new one using createTrans( ) in the AppLogic class (deprecated).

Tip

If an error occurs before the commit operation succeeds, the database server usually rolls back the transaction automatically.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
IResultSet rs;

// Update User

rs = updUserPQuery.execute(0, userValList, updCustTrans, null);

if (rs == null)

{

   updCustTrans.rollback();
```

```
    return handleOBSystemError("Could not update User.");
}
```

Related Topics

createTrans( ) in the AppLogic class (deprecated)

addRow( ), updateRow( ), and deleteRow( ) in the ITable Interface (deprecated)

IValList Interface *(deprecated)*

IValList is deprecated and is provided for backward compatibility only. New Java applications should use the standard servlet-JSP programming model.

For information about replacing IValList functionality in existing applications, see the *Migration Guide*.

An IValList represents a collection of GXVAL objects. This collection is not a sequential list, but an unordered set of GXVAL objects with no implied sequence or progression.

For iPlanet Application Server-enabled AppLogics, input arguments and output value(s) are stored in IValList objects. Every request to an AppLogic passes a list of input arguments, and every result from an AppLogic returns a list of output values. The AppLogic class (deprecated) defines two member variables, valIn and valOut, to contain the input arguments and output values, respectively, of AppLogic execution.

In an IValList, values and objects are mapped to keys. The key name is the name of a GXVAL object. AppLogic code refers to GXVAL object in the IValList by its key name. Key names are unique within each IValList object.

The IValList interface provides methods for adding, retrieving, removing, and counting GXVAL objects in the IValList instance. Using methods in the IValList interface, the AppLogic can test for input arguments and modify their contents for output values.

Keys may be passed to the AppLogic as a request from an HTML document or from another AppLogic module. In an HTML form, keys are often the field names defined in the form. In this way, the AppLogic can easily identify expected, common, or “well-known” keys, and the AppLogic can ignore irrelevant parameters.

For example, an AppLogic named getLogin might prompt users for their username and login, then pass this information, identified as “username” and “password”, to other AppLogics for processing. An AppLogic named validateLogin could retrieve the input parameters, find the values associated with the well-known keys “username” and “password”, then take action based on the data that the user entered (testing for its existence, performing a range or length check, looking up the combination in a password table, and so on).

To create an instance of the IValList interface, use the GX.CreateValList( ) method.

Package

com.kivasoft

Methods

Method	Description
count( )	Returns the number of GXVAL objects in the IValList.
getNextKey( )	Retrieves the key name of the next GXVAL object in the IValList.
getVal( )	Copies the specified GXVAL object from the IValList.
getValBLOB( )	Returns the specified BLOB object.
getValBLOBSize( )	Returns the size of a BLOB IValList object.
getValByRef( )	Gets the specified GXVAL object in the IValList.
getValInt( )	Retrieves an integer value from the specified GXVAL object in the IValList.
getValString( )	Retrieves a string value from the specified GXVAL object in the IValList.
removeVal( )	Removes the specified GXVAL object from the IValList.
resetPosition( )	Resets the iterator position to the “first” GXVAL object in the IValList.
setVal( )	Adds a GXVAL object to the IValList, or overwrites an existing one.
setValBLOB( )	Adds a BLOB object to the IValList object.
setValByRef( )	Adds a GXVAL object to the IValList, or overwrites an existing one.
setValInt( )	Adds a GXVAL object of type integer to the IValList, or overwrites an existing one.

Method	Description
setValString( )	Adds a GXVAL object of type string to the IValList, or overwrites an existing one.

Related Topics

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

count( )

Returns the number of GXVAL objects in the IValList .

Syntax

```
public int count()
```

Usage

When the contents of an IValList are unknown, an AppLogic can iterate through each GXVAL object to test, retrieve, and update information. Use count( ) to determine the maximum number of iterations needed to go completely through the IValList.

Rule

Do not add or remove GXVAL objects to or from the IValList when iterating through the IValList.

Tips

- Use count( ) in conjunction with getNextKey( ) and resetPosition( ) to iterate through the IValList.
- Adding or deleting GXVAL objects changes the number of objects in a IValList. Be sure to update the GXVAL object count after each add or delete operation.

Return Value

An int value representing the number of GXVal objects in the IValList, or zero for failure (such as the IValList is empty).

Example

```
// Obtain the contents of the shopping basket

IValList basket=session.getBasketContents();

// Determine whether basket is empty
```

```
if(basket.count()==0)
{
   // If basket contains items, process template to show
   if(evalTemplate(EMPTY_HTML, (ITemplateData) null, (ITemplateMap)
    null) ==0)
      return result("");
   else
// If basket is empty, just return message
   return result("Failed to Generate HTML");
}
```

Related Topics

getNextKey( )

resetPosition( )

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

getNextKey( )

Retrieves the key name of the next GXVAL object in the IValList.

Syntax

```
public String getNextKey()
```

Usage

When the contents of a IValList are unknown, the AppLogic can iterate through each GXVAL object and retrieve its key name. The AppLogic can then take action based on this information, or use the key name in operations that retrieve, update, or remove GXVAL objects in the IValList list.

Rule

Do not add or remove GXVAL objects to or from the IValList when iterating through the IValList.

Tip

Use getNextKey( ) in conjunction with count( ) and resetPosition( ) to iterate through the IValList.

Return Value

String representing the next retrieved key in the IValList, or null for failure (such as if no other keys exist).

Example

```
// Returns the contents of the shopping basket
public synchronized IValList getBasketContents()
{
   IValList list = getValList();
   IValList ret = GX.CreateValList();
   // Iterate through the contents of the shopping basket
   list.resetPosition();
   String key;
   while((key=list.getNextKey())!=null)
   {
      // Is this a desired key?
      if(key.startsWith(PROD))
      {
         // Obtain the product number from the item
         String prod = key.substring(PROD.length(),
             key.length());
         // Set the return IValList appropriately
         ret.setVal(prod, list.getVal(key));
      }
   }
   ret.resetPosition();
   return ret;
```

Related Topics

count( )

resetPosition( )

execute( ) in the AppLogic class (deprecated)

getVal( )

Copies the specified GXVAL object from the IValList.

Syntax

```
public GXVAL getVal(
   String pKey)
```

pKey. Key name of the GXVAL object to copy from the IValList.

Usage

Use getVal( ) if the data type of the GXVAL object is not known. Use getValString( ) instead for string objects, getValInt( ) for integer objects, and getValBLOB( ) for BLOB objects.

Rule

The specified key name must currently exist in the IValList.

Return Value

New GXVAL object representing a copy of the GXVAL object specified by pKey, or null for failure (such as invalid keyname).

Example

```
// Returns the contents of the shopping basket
public synchronized IValList getBasketContents()
{
   IValList list = getValList();
   IValList ret = GX.CreateValList();
   // Iterate through the contents of the shopping basket
   list.resetPosition();
   String key;
   while((key=list.getNextKey())!=null)
   {
      // Is this a desired key?
      if(key.startsWith(PROD))
      {
         // Obtain the product number from the item
         String prod=key.substring(PROD.length(),key.length());
```

```
            // Set the return IValList appropriately
            ret.setVal(prod, list.getVal(key));
         }
      }
      ret.resetPosition();
      return ret;
}
```

Related Topics

getValBLOB( ), getValInt( ), and getValString( ) in the IValList Interface (deprecated)

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

getValBLOB( )

Returns a specified BLOB object from the IValList.

Syntax

```
public byte[] getValBLOB(
   String pKey)
```

pKey. Key name of the GXVAL object that contains the BLOB value to retrieve.

Usage

Use getValBLOB( ) when the type of a GXVAL object is a BLOB, but its value is not known and needed for subsequent operations. Use getValString( ) instead for string objects and getValInt( ) for integer objects. If the type of the GXVAL object is not known, use getVal( ).

Rule

The data type must be TEXT, BINARY, VARBINARY, or database equivalent.

Return Value

A byte array for success, or null for failure.

Related Topics

getValBLOBSize( ) and setValBLOB( ) in the IValList Interface (deprecated)

getValBLOBSize( )

Returns the size of a specified BLOB object in the IValList.

Syntax

```
public int getValBLOBSize(
   String pKey)
```

pKey. Key name of the GXVAL object that contains the BLOB.

Usage

BLOB objects can be large. If you want to determine the size of a BLOB object before retrieving it, use getValBLOBSize( ).

Rule

The data type must be TEXT, BINARY, VARBINARY, or database equivalent.

Return Value

An integer value for success, or zero for failure (such as datatype mismatch).

Related Topics

getValBLOB( )

getValByRef( )

Gets the specified GXVAL object from the IValList.

Syntax

```
public GXVAL getValByRef(
   string pKey)
```

pKey. Key name of the GXVAL object to get from the IValList.

Usage

Use getValByRef( ) if the data type of the GXVAL object is not known, or if iterating through an IValList to get each GXVAL object. Use getValString( ) instead for string objects, getValInt( ) for integer objects, and getValBLOB( ) for BLOB objects.

Return Value

New GXVAL object representing a copy of the GXVAL object specified by pKey, or null for failure (such as invalid keyname).

Related Topics

getValBLOB( ), getValInt( ), and getValString( ) in the IValList Interface (deprecated)

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

getValInt( )

Retrieves an integer value from the specified GXVAL object in the IValList.

Syntax

```
public int getValInt(
   String pKey)
```

pKey. Key name of the GXVAL object from which to retrieve the integer value.

Usage

Use getValInt( ) if the data type of the GXVAL object is known to be an integer. Otherwise, use getValString( ) instead for string objects, getValBLOB( ) for BLOB objects, or getVal( ) for objects of other types.

Rules

- The specified key name must currently exist in the IValList.
- The data type of the specified GXVAL object must map to the enum value GXVT_I4.

Return Value

An int value representing the value of the specified GXVAL object, or null for failure (such as invalid key name or data type mismatch).

Example

```
// Obtains the User ID from the Session's IValList

public int getUserId()

{

   return getValList().getValInt("UserId");

}
```

Related Topics

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

getValString( )

Retrieves a string value from the specified GXVAL object in the IValList.

Syntax

```
public String getValString(
   String pKey)
```

pKey. Key name of the GXVAL object from which to retrieve the string value.

Usage

Use getValString( ) when the data type of the GXVAL object is known to be a string. Otherwise, use getValInt( ) instead for integer objects, getValBLOB( ) for BLOB objects, or getVal( ) for objects of other types.

Rules

- The specified key name must currently exist in the IValList.
- The data type of the specified GXVAL object must map to the enum value GXVT_LPSTR.

Tips

- An AppLogic can manipulate this retrieved value using member methods in the String class in the `java.lang` package.

Return Value

A String representing the value of the GXVAL object, or null for failure (such as invalid key name or data type mismatch).

Example

```
// Obtain security information from the valIn IValList
String nameParm = valIn.getValString("username");
String passwordParm = valIn.getValString("password");
String aclStr = valIn.getValString("accessControlLevel");

// Obtain the credit card info from the form
String cardStr=valIn.getValString("CreditCard");
String number=valIn.getValString("CreditCardNumber");
String month=valIn.getValString("ExpireMonth");
```

```
String year=valIn.getValString("ExpireYear");
```

Related Topics

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

removeVal( )

Removes the specified GXVAL object from the IValList.

Syntax

```
public int removeVal(
   String pKey)
```

pKey. Key name of the GXVAL object to remove from the IValList.

Usage

Use removeVal( ) to delete a GXVAL object that is no longer needed in the IValList . For example, if the AppLogic contains overloaded methods, you might want to remove a GXVAL object to ensure that the proper method is executed.

Rules

- The specified key name must currently exist in the IValList.
- Do not remove GXVAL objects from the IValList when iterating through the IValList.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Empties the contents of the shopping basket
public synchronized void emptyBasket(Agent agent)
{
   IValList list = getValList();
   while(true)
   {
      // Iterate over the contents
      list.resetPosition();
      String key;
```

```
      boolean repeat=false;
      while((key=list.getNextKey())!=null)
      {
         // Is this a desired key?
         if(key.startsWith(PROD))
         {
            // Remove the Val object from the shopping cart
            list.removeVal(key); repeat=true; break;
         }
      }
      if(!repeat) break;
   }
```

Related Topics

setVal( ) and resetPosition( ) in the IValList Interface (deprecated)

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

resetPosition( )

Resets the iterator position to the “first” GXVAL object in the IValList.

Syntax

```
public int resetPosition()
```

Usage

When the contents of an IValList are unknown, the AppLogic can iterate through each GXVAL object and retrieve its key name. Before iterating through the IValList , the AppLogic needs to call resetPosition( ) once to ensure that iteration begins at the “first” GXVAL object in the IValList .

Rule

Do not add or remove GXVAL objects to or from the IValList when iterating through the IValList.

Tips

- The first GXVAL object is not necessarily the first one added to the IValList.

- Use resetPosition( ) in conjunction with count( ) and getNextKey( ) to iterate through the IValList.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Returns the contents of the shopping basket
public synchronized IValList getBasketContents()
{
   IValList list=getValList();
   IValList ret = GX.CreateValList();
   // Iterate through the contents of the shopping basket
   list.resetPosition();
   String key;
   while((key=list.getNextKey())!=null)
   {
      // Is this a desired key?
      if(key.startsWith(PROD))
      {
         // Obtain the product number from the item
         String prod = key.substring(PROD.length(),
                           key.length());
         // Set the return IValList appropriately
         ret.setVal(prod, list.getVal(key));
      }
   }
   ret.resetPosition();
   return ret;
}
```

Related Topics

count( ) and getNextKey( ) in the IValList Interface (deprecated)

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

setVal( )

Copies a GXVAL object to the IValList .

Syntax

```
public int setVal(
   String pKey,
   GXVAL pVal)
```

pKey. Key name of the GXVAL object to add to the IValList.

pVal. The GXVAL object, identified by pKey, to add to the IValList.

Usage

Use setVal( ) to add an existing GXVAL object to the IValList. If a GXVAL object with the same key name already exists, setVal( ) overwrites it with the new one.

Rule

Do not add new GXVAL objects to the IValList when iterating through the IValList.

Tip

To add a new GXVAL object of type integer, string, or BLOB to the IValList, use setValInt( ), setValString( ), or setValBLOB( ), respectively.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Returns the contents of the shopping basket
public synchronized IValList getBasketContents()
{
   IValList list=getValList();
   IValList ret = GX.CreateValList();
   // Iterate through the contents of the shopping basket
   list.resetPosition();
   String key;
   while((key=list.getNextKey())!=null)
   {
```

```
      // Is this a desired key?
      if(key.startsWith(PROD))
      {
         // Obtain the product number from the item
         String prod = key.substring(PROD.length(),
                           key.length());
         // Set the return IValList appropriately
         ret.setVal(prod, list.getVal(key));
      }
   }
   ret.resetPosition();
   return ret;
}
```

Related Topics

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

setValBLOB( )

Adds a BLOB object to the IValList

Syntax

```
public int setValBLOB(
   String pKey,
   byte[] pBuff,
   int nBuffLen)
```

pKey. Key name of the GXVAL object to add to the IValList.

pBuff. The value of the BLOB object to add to the IValList.

nBuffLen. Number of bytes to set for the byte array. The first nBuffLen bytes in the array pBuff hold the value.

Usage

Use setValBLOB( ) to add a GXVAL object that contains a BLOB value to the IValList. If a GXVAL object with the same key name already exists, setValBLOB( ) overwrites it with the new one.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
getValBLOB( )

getValBLOBSize( )

setValByRef( )

Copies a GXVAL object to the IValList .

Syntax
```
public int setValByRef(
   String pKey,
   GXVAL pVal)
```

pKey. Key name of the GXVAL object to add to the IValList.

pVal. The GXVAL object, identified by pKey, to add to the IValList.

Usage
Use setValByRef( ) to add an existing GXVAL object to the IGXValList. If a GXVAL object with the same key name already exists, setValByRef( ) overwrites it with the new one.

Rule
Do not add new GXVAL objects to the IValList when iterating through the IValList.

Tip
To add a new GXVAL object of type integer, string, or BLOB to the IValList, use setValInt( ), setValString( ), or setValBLOB( ), respectively.

Return Value
GXE.SUCCESS if the method succeeds.

Related Topics
valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

setValInt( )

Adds a GXVAL object of type integer to the IValList.

Syntax

```
public int setValInt(
    String pKey,
    int nVal)
```

pKey. Key name of the GXVAL object to create or overwrite.

nVal. The integer value to assign to the GXVAL object identified by pKey.

Usage

Use setValInt( ) to add a GXVAL object of type integer to the IValList. If a GXVAL object with the same key name already exists, setValInt( ) overwrites it with the new one.

Rules

When iterating through existing GXVAL objects in the IValList, do not add new GXVAL objects to the IValList.

Tips

To add a new GXVAL object of type string or BLOB to the IValList, use setValString( ) or setValBLOB( ), respectively.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Specifies the User ID for the Session's IValList

public void setUserId(int uid, Agent agent)

{

   getValList().setValInt("UserId", uid);

   agent.saveSession();

}
```

Related Topics

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

setValString( )

Adds a GXVAL object of type string to the IValList.

Syntax

```
public int SetValString(
   String pKey,
   String val)
```

pKey. Key name of the GXVAL object to create or overwrite.

val. The string value to assign to the GXVAL object identified by pKey.

Usage

Use setValString( ) to add a GXVAL object of type string to the IValList. If a GXVAL object with the same key name already exists, setValString( ) overwrites it with the new one.

Rules

When iterating through existing GXVAL objects in the IValList, do not add new GXVAL objects to the IValList.

Tips

To add a new GXVAL object of type integer or BLOB to the IValList, use setValInt( ) or setValBLOB( ), respectively.

Return Value

GXE.SUCCESS if the method succeeds.

Example

```
// Create and IValList and set string values

IValList custValList = GX.CreateValList();

custValList.setValString(":ssn", ssn);

custValList.setValString(":userName", userName);
```

Related Topics

valIn and valOut in the AppLogic class (deprecated)

execute( ) in the AppLogic class (deprecated)

Return Codes

Many methods in the iPlanet Application Server Foundation Class Library return error codes. These generic error codes are defined as static values in the GXE class. When testing for a returned error code, prefix the error code with the class name, as shown in the following example:

```
if (hr==GXE.SUCCESS)
```

The following table lists the error codes defined in the GXE class:

Error code	Value
SUCCESS	0x80240000
ERROR	0x80240001
INVALID_ARG	0x80240002
INVALID_INTERFACE	0x80240003
NOT_SUPPORTED	0x80240004
EOF	0x80240005
READ_FAILED	0x80240006
WRITE_FAILED	0x80240007
ALLOC_FAILED	0x80240008
INVALID_NAME	0x80240009
INVALID_EXPR	0x8024000a
INVALID_INDEX	0x8024000b
TOO_SMALL	0x8024000c
FAIL	0x8024000d
NOINTERFACE	0x8024000e

Error code	Value
MEM_ALLOC_FAILED	0x8024000f

Index

E

F

G

H

I

L

M

T

U

V

W