SYBASE®

API Reference Manual

# EAServer

6.0

DOCUMENT ID: DC38037-01-0600-01

LAST REVISED: July 2006



Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

# Contents

# About This Book

This book, the *EAServer API Reference Manual*, contains reference pages for EAServer proprietary Java classes, C++ classes, ActiveX interfaces, and C routines. EAServer also supports many standard Java 2 Enterprise Edition (J2EE) and CORBA APIs. For information on these, see:

- The *Enterprise JavaBeans User's Guide*.
- The *CORBA Components Guide*.
- The relevant standards document for API reference information. For J2EE standards documents, please see the Sun Microsystems J2EE Web pages at http://java.sun.com/. For CORBA standards documentation, please see the Object Management Group (OMG) Web site at http://www.omg.org/.

## Audience

This book is written as a reference for developers of EAServer applications. Developers should know their development language and programming tools.

## How to use this book

Chapter 1, "Java Classes and Interfaces," documents EAServer's Java classes and interfaces. You will need this information to implement Java components or Java clients.

Chapter 2, "C Routines Reference," documents EAServer's C library routines. You will need this information to implement C components.

Appendix A, "Deprecated Java Classes and Interfaces," documents Java classes and interfaces supported solely for backward compatibility.

## Related documents

**Core EAServer documentation** The core EAServer documents are available in HTML and PDF format in your EAServer software installation and on the SyBooks™ CD.

*What's New in EAServer 6.0* summarizes new functionality in this version.

The *EAServer API Reference Manual* (this book) contains reference pages for proprietary EAServer Java classes and C routines.

The *EAServer Automated Configuration Guide* explains how to use Ant-based configuration scripts to:

- Define and configure entities, such as EJB modules, Web applications, data sources, and servers
- Perform administrative and deployment tasks

The *EAServer CORBA Components Guide* explains how to:

- Create, deploy, and configure CORBA and PowerBuilder™ components and component-based applications
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Enterprise JavaBeans User's Guide* describes how to:

- Configure and deploy EJB modules
- Develop EJB clients, and create and configure EJB providers
- Create and configure applications clients
- Run the EJB tutorial

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer Java Message Service User's Guide* describes how to create Java Message Service (JMS) clients and components to send, publish, and receive JMS messages.

The *EAServer Migration Guide* contains information about migrating EAServer 5.*x* resources and entities to an EAServer 6.0 installation.

The *EAServer Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

The *EAServer Security Administration and Programming Guide* explains how to:

- Understand the EAServer security architecture
- Configure role-based security for components and Web applications
- Configure SSL certificate-based security for client connections
- Implement custom security services for authentication, authorization, and role membership evaluation
- Implement secure HTTP and IIOP client applications
- Deploy client applications that connect through Internet proxies and firewalls

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured server and manage it with the Sybase Management Console
- Create, configure, and start new application servers
- Define database types and data sources
- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components
- Automate administration and monitoring tasks with command line tools

The *EAServer Web Application Programming Guide* explains how to create, deploy, and configure Web applications, Java servlets, and JavaServer Pages.

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_5.2.eastg/html/eastg/title.htm.

**jConnect for JDBC documents** EAServer includes the jConnect™ for JDBC™ 6.0.5 driver to allow JDBC access to Sybase database servers and gateways. The *jConnect for JDBC 6.0.5 Programmer's Reference* is available on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.jconnjdbc_6.05.prjdbc/html/prjdbc/title.htm&toc=/com.sybase.help.jconnjdbc_6.05/toc.xml.

**Sybase Software Asset Management User's Guide** EAServer includes the Sybase Software Asset Management license manager for managing and tracking your Sybase software license deployments. The *Sybase Software Asset Management User's Guide* is available on the Getting Started CD and in the EAServer 6.0 collection on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_6.0/title.htm.

**Conventions**

The formatting conventions used in this manual are:

| Formatting example | To indicate |
|---|---|
| commands and methods | When used in descriptive text, this font indicates keywords such as:<br>• Command names used in descriptive text<br>• C++ and Java method or class names used in descriptive text<br>• Java package names used in descriptive text<br>• Property names in the raw format, as when using Ant or jagtool to configure applications rather than the Management Console |
| *variable*, *package*, or *component* | Italic font indicates:<br>• Program variables, such as *myCounter*<br>• Parts of input text that must be substituted, for example:<br>`Server.log`<br>• File names<br>• Names of components, EAServer packages, and other entities that are registered in the EAServer naming service |
| File \| Save | Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File \| Save indicates "select Save from the File menu." |
| `package 1` | Monospace font indicates:<br>• Information that you enter in the Management Console, a command line, or as program text<br>• Example program fragments<br>• Example output fragments |

**Other sources of information**

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com/nav/base.do.

**Sybase certifications on the Web**

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

**Sybase EBFs and software maintenance**

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

**Accessibility features**

EAServer has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in Eclipse help formats, which you can navigate using a screen reader.

The Web console supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Management Console Overview," in the *EAServer System Administration Guide*.

The Web Services Toolkit plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired, or have other special needs. For information about these features see the Eclipse help:

1 Start Eclipse.

2 Select Help | Help Contents.

3 Enter `Accessibility` in the Search dialog box.

4 Select Accessible User Interfaces or Accessibility Features for Eclipse.

---

**Note** You may need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

---

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

**If you need help**

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

# Java Classes and Interfaces

## Package index

### com.sybase.CORBA.jdbc11

For use in classes that will be run in a JDK-1.1-compatible Java virtual machine. Provides classes for converting between EAServer's predefined IDL datatypes and the core Java language objects:

- IDL – Provides methods to convert core Java datatypes to EAServer's predefined CORBA IDL datatypes.
- IdlResultSet – Implements the JServerResultSet interface, allowing you to construct TabularResults.ResultSet instances for component methods that return row results.
- SQL – Provides methods to convert EAServer's predefined CORBA IDL datatypes to core Java datatypes.

---

**Note** Open source implementations of the TabularResults classes are available on the EAServer CodeXchange pages at http://easerver.codexchange.sybase.com/.

---

### com.sybase.jaguar.jcm

Classes and interfaces for managing cached JDBC connections in server-side Java code:

- com.sybase.jaguar.jcm.JCM class – Provides access to JDBC data sources.
- com.sybase.jaguar.jcm.JCMCache class – Manages a pool of JDBC connections to a third-tier database server.
- com.sybase.jaguar.jcm.JConnectionNotFoundException class – Exception thrown when no connections are available.

## com.sybase.jaguar.server

Utility classes used in server-side Java code:

- com.sybase.jaguar.server.Jaguar class – Provides utility methods for use in server-side Java code.
- com.sybase.jaguar.server.JContext class – Instantiates objects that are used to send result sets from a Java component method and provides a method to retrieve rows from a java.sql.ResultSet and forward them to the client.

## com.sybase.jaguar.sql

Interfaces for objects that construct and send row results from a Java server component to the client:

- com.sybase.jaguar.sql.JServerResultSet interface – Provides methods to return result rows to a client application. JServerResultSet is similar to the java.sql.ResultSet interface, which is used to retrieve result rows from a server.
- com.sybase.jaguar.sql.JServerResultSetMetaData interface – Provides methods for describing the metadata of a result set. Metadata specifies the number of columns in each row as well as the datatype, format, nullability, and so forth for each column.

## com.sybase.jaguar.util

Utility classes that are used in both server-side and client side Java code:

- com.sybase.jaguar.util.JException class – JException is the generic exception that is thrown by methods in the EAServer classes or in generated client stub classes.

# com.sybase.CORBA.jdbc11.IDL class

Description

package com.sybase.CORBA.jdbc11;

public abstract class IDL

Provides methods to convert core Java datatypes to EAServer's predefined CORBA IDL datatypes.

Constructors

None. All methods are static.

Methods

- getDate(java.sql.Date) – Converts a java.sql.Date object to an equivalent MJD::Date CORBA IDL object.
- getDecimal(java.math.BigDecimal) – Converts a BigDecimal object to an equivalent BCD::Decimal CORBA IDL object.
- getMoney(java.math.BigDecimal) – Converts a BigDecimal object to an equivalent BCD::Money CORBA IDL object.
- getResultSet(java.sql.ResultSet) – Converts a java.sql.ResultSet object to an equivalent TabularResults::ResultSet CORBA IDL object.
- getTime(java.sql.Time) – Converts a java.sql.Time object to an equivalent MJD::Time CORBA IDL object.
- getTimestamp(java.sql.Timestamp) – Converts a java.sql.Timestamp object to an equivalent MJD::Timestamp CORBA IDL object.

See also

com.sybase.CORBA.jdbc11.SQL class

## IDL.getDate(java.sql.Date)

Description

Converts a java.sql.Date object to an equivalent MJD::Date CORBA IDL object.

Syntax

| Package | com.sybase.CORBA.jdbc11 |
|---|---|
| Class | IDL |

public static MJD.Date getDate(java.sql.Date value)

Parameters

*value*

A java.sql.Date value to be converted.

Return value

The value converted to an equivalent CORBA IDL MJD::Date value.

See also

getTime(java.sql.Time), getTimestamp(java.sql.Timestamp), SQL.getDate(MJD.Date)

## IDL.getDecimal(java.math.BigDecimal)

Description

Converts a BigDecimal object to an equivalent BCD::Decimal CORBA IDL object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |
| Class | IDL |

```
public static BCD.Decimal
    getDecimal(java.math.BigDecimal value)
    throws org.omg.CORBA.DATA_CONVERSION
```

Parameters

*value*

A java.math.BigDecimal value to be converted.

Return value

The value converted to an equivalent CORBA IDL BCD::Decimal value.

See also

getMoney(java.math.BigDecimal), SQL.getBigDecimal(BCD.Decimal)

## IDL.getMoney(java.math.BigDecimal)

Description

Converts a BigDecimal object to an equivalent BCD::Money CORBA IDL object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |
| Class | IDL |

```
public static BCD.Money getMoney(
     java.math.BigDecimal value)
    throws org.omg.CORBA.DATA_CONVERSION
```

Parameters

*value*

A java.math.BigDecimal value to be converted.

Return value

The value converted to an equivalent CORBA IDL BCD::Money value.

See also

getDecimal(java.math.BigDecimal), SQL.getBigDecimal(BCD.Money)

## IDL.getResultSet(java.sql.ResultSet)

Description

Converts a java.sql.ResultSet object to an equivalent TabularResults::ResultSet CORBA IDL object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |
| Class | IDL |

```
public static MJD.ResultSet
    getResultSet( java.sql.ResultSet rs)
```

Parameters

*rs*

A java.sql.ResultSet value to be converted.

Return value

The value converted to an equivalent CORBA IDL TabularResults::ResultSet value.

See also

SQL.getResultSet(TabularResults.ResultSet)

## IDL.getTime(java.sql.Time)

Description

Converts a java.sql.Time object to an equivalent MJD::Time CORBA IDL object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |
| Class | IDL |

```
public static MJD.Time getTime(java.sql.Time value)
```

Parameters

*value*

A java.sql.Time value to be converted.

Return value

The value converted to an equivalent CORBA IDL MJD::Time value.

See also

getDate(java.sql.Date), getTimestamp(java.sql.Timestamp), SQL.getTime(MJD.Time)

## IDL.getTimestamp(java.sql.Timestamp)

Description

Converts a java.sql.Timestamp object to an equivalent MJD::Timestamp CORBA IDL object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |
| Class | IDL |

```
public static MJD.Timestamp
    getTimestamp( java.sql.Timestamp value)
```

Parameters

*value*
A java.sql.Timestamp value to be converted.

Return value

The value converted to an equivalent CORBA IDL MJD::Timestamp value.

See also

getDate(java.sql.Date), getTime(java.sql.Time), SQL.getTimestamp(MJD.Timestamp)

# com.sybase.CORBA.jdbc11.IdlResultSet

Description

package com.sybase.CORBA.jdbc11;

public class IdlResultSet
  extends java.lang.Object
  implements jaguar.sql.JServerResultSet;

Implements the JServerResultSet interface, allowing you to construct TabularResults.ResultSet instances for component methods that return row results.

Component methods that return row results to clients return TabularResults.ResultSet or TabularResults.ResultSet[]. IdlResultSet allows you to create instances of these types using the JDBC style JServerResultSet interfaces.

For documentation of the TabularResults IDL types, see the generated Interface Repository documentation at ../../ir/TabularResults.html.

To return a single result set, initialize the rows and columns using the JServerResultSetMetaData and JServerResultSet methods, then convert to a TabularResults.ResultSet instance as shown in this code fragment:

```
JServerResultSetMetaData jsrs;
... define column formats ...
IdlResultSet irs = new IdlResultSet(jsrsmd);
... define row data using JServerResultSet methods ...
return irs.getResultSet();
```

To return multiple result sets, build an array of TabularResults.ResultSet instances, as follows:

1 Declare a java.util.Vector instance:

```
java.util.Vector vector = new Vector();
```

2 Initialize each IdlResultSet instance as described above, then add it to the vector:

```
    vector.addElement(irs.getResultSet());
```

3 When done, convert the vector to an array to be returned by the method:

```
TabularResults.ResultSet[] array =
    new TabularResults.ResultSet[vector.size()];
vector.copyInto(array);
return array;
```

Constructors

- IdlResultSet(java.sql.ResultSetMetaData) – Construct an instance using the column formats specified by a JServerResultSetMetaData instance. You can add rows to the instance using the JServerResultSet methods.
- IdlResultSet(java.sql.ResultSet) – Construct an instance by reading the rows from the supplied ResultSet.

Methods

- getResultSet() – Translate the contents of this instance into TabularResults.ResultSet instance.

See also

com.sybase.jaguar.sql.JServerResultSet interface, com.sybase.jaguar.sql.JServerResultSetMetaData interface

# com.sybase.CORBA.jdbc11.SQL class

Description

package com.sybase.CORBA.jdbc11;

public abstract class SQL

Provides methods to convert EAServer's predefined CORBA IDL datatypes to core Java datatypes.

Constructors

None. All methods are static.

Methods

- getBigDecimal(BCD.Decimal) – Converts a BCD::Decimal CORBA IDL object to an equivalent java.math.BigDecimal.
- getBigDecimal(BCD.Money) – Converts a BCD::Money CORBA IDL object to an equivalent java.math.BigDecimal.
- getDate(MJD.Date) – Converts an MJD::Date CORBA IDL object to an equivalent java.sql.Date object.
- getResultSet(TabularResults.ResultSet) – Converts a TabularResults::ResultSet CORBA IDL object to an equivalent java.sql.ResultSet object.
- getTime(MJD.Time) – Converts an MJD::Time CORBA IDL object to an equivalent java.sql.Time object.

- getTimestamp(MJD.Timestamp) – Converts an MJD::Timestamp CORBA IDL object to an equivalent java.sql.Timestamp object.

See also com.sybase.CORBA.jdbc11.IDL class

## SQL.getBigDecimal(BCD.Decimal)

Description Converts a BCD::Decimal CORBA IDL object to an equivalent java.math.BigDecimal.

Syntax

| Package | com.sybase.CORBA.jdbc11 |
|---|---|
| Class | SQL |

```
public static java.math.BigDecimal
    getBigDecimal(BCD.Decimal value)
```

Parameters *value*
A BCD.Decimal value to be converted.

Return value The value converted to an equivalent java.math.BigDecimal value.

See also getBigDecimal(BCD.Decimal), getBigDecimal(BCD.Money), IDL.getDecimal(java.math.BigDecimal)

## SQL.getBigDecimal(BCD.Money)

Description Converts a BCD::Money CORBA IDL object to an equivalent java.math.BigDecimal.

Syntax

| Package | com.sybase.CORBA.jdbc11 |
|---|---|
| Class | SQL |

```
public static java.math.BigDecimal
    getBigDecimal(BCD.Money value)
```

Parameters *value*
A BCD.Money value to be converted.

Return value The value converted to an equivalent java.math.BigDecimal value.

See also getBigDecimal(BCD.Decimal), IDL.getMoney(java.math.BigDecimal)

## SQL.getDate(MJD.Date)

Description: Converts an MJD::Date CORBA IDL object to an equivalent java.sql.Date object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |
| Class | SQL |

```
public static java.sql.Date getDate(MJD.Date value)
```

Parameters

*value*
An MJD::Date value to be converted.

Return value: The value converted to an equivalent java.sql.Date value.

See also: getTime(MJD.Time), getTimestamp(MJD.Timestamp), IDL.getDate(java.sql.Date)

## SQL.getResultSet(TabularResults.ResultSet)

Description: Converts a TabularResults::ResultSet CORBA IDL object to an equivalent java.sql.ResultSet object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |
| Class | SQL |

```
public static java.sql.ResultSet
    getResultSet(TabularResults.ResultSet rs)
```

Parameters

*rs*
A TabularResults.ResultSet object to be converted.

Return value: The value converted to an equivalent java.sql.ResultSet value.

See also: IDL.getResultSet(java.sql.ResultSet)

## SQL.getTime(MJD.Time)

Description: Converts an MJD::Time CORBA IDL object to an equivalent java.sql.Time object.

Syntax

| | |
|---|---|
| Package | com.sybase.CORBA.jdbc11 |

| Class | SQL |
|---|---|

public static java.sql.Time getTime(MJD.Time value)

Parameters

*value*
An MJD.Time value to be converted.

Return value

The value converted to an equivalent java.sql.Time value.

See also

getDate(MJD.Date), getTimestamp(MJD.Timestamp), IDL.getTime(java.sql.Time)

## SQL.getTimestamp(MJD.Timestamp)

Description

Converts an MJD::Timestamp CORBA IDL object to an equivalent java.sql.Timestamp object.

Syntax

| Package | com.sybase.CORBA.jdbc11 |
|---|---|
| Class | SQL |

public static java.sql.Timestamp
getTimestamp(MJD.Timestamp value)

Parameters

*value*
An MJD.Timestamp value to be converted.

Return value

The value converted to an equivalent java.sql.Timestamp value.

See also

getDate(MJD.Date), getTime(MJD.Time), IDL.getTimestamp(java.sql.Timestamp)

# com.sybase.jaguar.jcm.JCM class

Description

package com.sybase.jaguar.jcm;
public class JCM extends Object

Provides access to JDBC data sources.

Constructors

None. All methods are static.

Methods

- byNameAllowed(String) – Determines if a data source can be retrieved by calling getCacheByName(String).

- getCache(String, String, String) – Returns a reference to a data source with matching values for the specified user name, password, and server name.
- getCacheByName(String) – Returns a reference to the data source with the given name.

## JCM.byNameAllowed(String)

Description

Determines if a data source can be retrieved by calling getCacheByName(String).

**Note** Beginning in EAServer 6.0, all data sources allow access by name. This method is provided for backward compatibility.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Interface | JCM |

```
public static boolean byNameAllowed
   (String name)
   throws JException
```

Parameters

*name*

The name of the data source of interest.

Return value

`true` if a data source is installed with the specified name, and the data source can be retrieved with JCM.getCacheByName(String); `false` otherwise.

Usage

The getCacheByName(String) method allows you to retrieve a data source by specifying only the data source name, rather than specifying values for the data source user name, password, and server name.

You can call byNameAllowed to determine whether by-name access is allowed for a specified data source.

See also

getCacheByName(String)

## JCM.getCache(String, String, String)

Description

Returns a reference to a data source with matching values for the specified user name, password, and server name.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Interface | JCM |

```
public static JCMCache getCache
  ( String user, String pwd, String server)
  throws JException
```

Parameters

*user*
The database user name associated with the data source.

*pwd*
The database password associated with the data source.

*server*
The database server name associated with the data source. The value should be a JDBC connection URL in the appropriate format for calls to java.sql.DriverManager.getConnection(String). The URL format depends on which JDBC driver the data source uses. See your JDBC driver documentation for more information.

Return value

A reference to a JCMCache instance with matching values for *user*, *pwd*, and *server*.

A JException exception is thrown if no data source with matching values exists.

Usage

The supplied values for *user*, *pwd*, and *server* must match the properties of an existing data source.

See also

Chapter 4, "Database Access," in the *EAServer System Administration Guide*

getCacheByName(String)

## JCM.getCacheByName(String)

Description

Returns a reference to the data source with the specified name.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Interface | JCM |

```
public static JCMCache getCacheByName
  ( String name)
  throws JException
```

Parameters

*name*
The name of the data source to be retrieved.

Return value

A reference to a JCMCache instance with a matching value for *name*.

A JException exception is thrown if:

- No data source is installed with the specified name.
- A matching data source is installed, but the data source properties forbid retrieval with this method. Use getCache(String, String, String) instead.

Usage

getCacheByName allows you to retrieve a data source by specifying only the data source name, rather than specifying values for the data source user name, password, and server name.

Using this method rather than getCache(String, String, String) removes the need to code database user names and passwords into your component source code. This method also allows you to change the data source user name, password, or server in the data source properties without requiring corresponding changes to your component source code.

---

**Note** Beginning in EAServer 6.0, all data sources allow access by name.

---

See also

Chapter 4, "Database Access," in the *EAServer System Administration Guide*

byNameAllowed(), getCache(String, String, String)

# com.sybase.jaguar.jcm.JCMCache class

Description

```
package com.sybase.jaguar.jcm;
public class JCMCache extends Object
```

Manages a pool of connections to a third-tier database server.

Constructors

None. Call JCM.getCache(String, String, String).

Fields

JCM_FORCE

```
public final static int JCM_FORCE
```

A value for the getConnection *flag* parameter.

JCM_NOWAIT

```
public final static int JCM_NOWAIT
```

A value for the getConnection *flag* parameter.

JCM_WAIT

public final static int JCM_WAIT

A value for the getConnection *flag* parameter.

Methods

- byNameAllowed() – Determines whether the data source can be retrieved by calling JCM.getCacheByName(String).
- dropConnection(Connection) – Drops a connection. The connection is closed and not released into the data source.
- getPoolSizeMax() – Retrieves the maximum number of connections that this data source can manage.
- getConlibName() – Returns the connectivity library (or interface) name for the data source.
- getConnection(int) – Obtains a connection handle from the data source.
- getProxyConnection(int, String) – Obtains a connection handle from the data source, specifying an alternate login name to set-proxy to.
- getName() – Retrieves the data source's name.
- getPassword() – Retrieves the password used by connections in the data source.
- getRemoteServerName() – Returns the remote server name used by connections in the data source.
- getUsername() – Retrieves the user name used by connections in the data source.
- releaseConnection(Connection) – Releases a connection to the data source for reuse.

## JCMCache.byNameAllowed()

Description

Determines whether the cache can be retrieved by calling JCM.getCacheByName(String).

**Note** Beginning in EAServer 6.0, all data sources allow access by name. This method is provided for backward compatibility.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

public boolean byNameAllowed()

Return value — `true` if the data source can be retrieved with JCM.getCacheByName(String), `false` otherwise.

See also — getName(), JCM.byNameAllowed(String), JCM.getCacheByName(String)

See Chapter 4, "Database Access," in the *EAServer System Administration Guide*

## JCMCache.dropConnection(Connection)

Description — Drops a connection. The connection is closed and not released into the cache.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public void dropConnection( Connection con)
   throws SQLException
```

Parameters

*con*
The java.sql.Connection instance to be dropped.

Usage — Use dropConnection() to close a connection when you do not want the connection returned to the data source. If necessary, future getConnection(int) calls will allocate new connections to replace any that have been dropped.

See also — getConnection(int), releaseConnection(Connection)

## JCMCache.getConlibName()

Description — Returns the connectivity library (or interface) name for the data source.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public String getConlibName()
```

Return value — "JDBC"

## JCMCache.getConnection(int)

Description — Obtains a connection handle from the data source.

Syntax

| Package | com.sybase.jaguar.jcm |
|---|---|
| Class | JCMCache |

```
public Connection getConnection(int flag)
   throws SQLException, JException,
   JConnectionNotFoundException
```

Parameters

*flag*

A symbolic value that specifies what should happen if the maximum number of connections have been allocated and are in use (that is, no connection is available in the data source). Allowable values are:

| Value | Behavior when no connection is available |
|---|---|
| JCM_NOWAIT | Throws JConnectionNotFoundException. |
| JCM_WAIT | Does not return until a cached connection is available. |
| JCM_FORCE | "Forces" open a new, uncached connection. The data source's maximum size is ignored. |

Return value

A java.sql.Connection instance from the data source. If the call specifies JCM_NOWAIT and no connections are available, the call throws a JConnectionNotFoundException instance.

Usage

getConnection(int) attempts to return a connection from the data source. data sources are maintained statically; a data source is initially empty when the server starts. Subsequent getConnection(int) calls allocate connections when necessary. releaseConnection(Connection) calls release control of a connection for later reuse.

Each data source has a maximum number of connections determined by the data source properties. (See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.) The *flag* parameter determines getConnection(int) behavior when the data source's maximum number of connections are in use. getPoolSizeMax() returns the data source's maximum number of connections.

For improved performance, connections should not be held any longer than necessary. As a general rule, methods that use a cached connection should release it with releaseConnection(Connection) before returning. This strategy minimizes contention by multiple components for a data source's connections.

See also

dropConnection(Connection), getPoolSizeMax(), releaseConnection(Connection)

## JCMCache.getPoolSizeMax()

Description: Retrieves the maximum number of connections that can be pooled in the data source.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public int getPoolSizeMax()
```

Return value: The data source size.

Usage: The size is specified the data source properties. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

See also: getPoolSizeMin()

## JCMCache.getPoolSizeMin()

Description: Retrieves the maximum number of connections that can be pooled in the data source.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public int getPoolSizeMax()
```

Return value: The data source size.

Usage: The size is specified the data source properties. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

See also: getPoolSizeMax()

## JCMCache.getProxyConnection(int, String)

Description: Obtains a connection handle from the data source, specifying an alternate login name to set-proxy to.

**Not all data sources support set-proxy**
Set-proxy support must be enabled in the data source properties before you can use this feature. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information. You must be connected to a database server, such as Adaptive Server Enterprise 11.5 or later, that supports the set session authorization command.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public Connection getProxyConnection(int flag, String proxy)
   throws SQLException, JException,
  JConnectionNotFoundException
```

Parameters

*flag*

A symbolic value that specifies what should happen if the maximum number of connections have been allocated and are in use (that is, no connection is available in the data source). Allowable values are:

| Value | Behavior when no connection is available |
|---|---|
| JCM_NOWAIT | Throws JConnectionNotFoundException. |
| JCM_WAIT | Does not return until a cached connection is available. |
| JCM_FORCE | "Forces" open a new, uncached connection. The data source's maximum size is ignored. |

*proxy*

The user name to set-proxy to.

Return value

A java.sql.Connection instance from the data source. If the call specifies JCM_NOWAIT and no connections are available, the call throws a JConnectionNotFoundException instance.

Usage

This method retrieves a cached connection, specifying an alternate login name to set-proxy to. Set-proxy support must be enabled in the data source properties. If support is enabled, connections retrieved from the data source with getConnection(int) set-proxy to the client user name. Call getProxyConnection(int, String) to specify a different user name to set-proxy to.

Other than the set-proxy behavior, getProxyConnection(int, String) is identical to getConnection(int).

See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for information on defining data sources and enabling set-proxy support.

For improved performance, connections should not be held any longer than necessary. As a general rule, methods that use a cached connection should release it with releaseConnection(Connection) before returning. This strategy minimizes contention by multiple components for a data source's connections.

See also dropConnection(Connection), getPoolSizeMax(), getConnection(int), releaseConnection(Connection)

## JCMCache.getName()

Description Retrieves the data source's name.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public String getName()
```

Return value The data source's name.

## JCMCache.getPassword()

Description Retrieves the password used by connections in the data source.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public String getPassword()
```

Return value The password.

See also getRemoteServerName(), getUsername()

## JCMCache.getRemoteServerName()

Description Retrieves the remote server name used by connections in the data source.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public String getRemoteServerName()
```

Return value — The remote server name.

See also — getPassword(), getUsername()

## JCMCache.getUserName()

Description — Retrieves the user name used by connections in the data source.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public String getUserName()
```

Return value — The user name.

See also — getPassword(), getRemoteServerName()

## JCMCache.releaseConnection(Connection)

Description — Releases a connection to the data source for reuse.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.jcm |
| Class | JCMCache |

```
public void releaseConnection( Connection con)
  throws SQLException
```

Parameters

*con*
The connection to release.

Usage — Released connections must be in a state that allows new queries to be issued.

The connection will be dropped (and not returned to the data source) if the data source has exceeded its maximum number of connections. The maximum number of connections can be exceeded if calls to getConnection(int) are issued with *flag* as JCM_FORCE. In this case, releaseConnection drops the excess connections.

Many JDBC programs do not explicitly clean up java.sql.Statement objects. Instead, they rely on the JDBC driver to clean up Statement objects when the connection is closed. This strategy does not work with cached connections: you must explicitly clean up Statement objects before releasing a connection back into the data source. To clean up Statement objects, call Statement.close() and set the Statement reference to `null`.

**Warning!** To prevent memory leaks, you must explicitly clean up a connection's Statement objects before releasing the connection back into the data source. Do not release a connection more than once.

See also getConnection(int), dropConnection(Connection)

## com.sybase.jaguar.jcm.JConnectionNotFoundException class

Description

```
package com.sybase.jaguar.jcm;
public class JConnectionNotFoundException
  extends JException;
```

Exception thrown by JCMCache.getConnection(int) to indicate that no connections are available in the data source. You must specify JCM_NOWAIT in order for the exception to be thrown.

Constructors Same as JException.

Methods Same as JException.

See also com.sybase.jaguar.util.JException class, java.sql.SQLException class

## com.sybase.jaguar.server.Jaguar class

Description

```
package com.sybase.jaguar.server;
public class Jaguar extends Object
```

Provides utility methods for use in server-side Java code.

Constructors None. All methods are static.

Methods

- getInstanceContext() – Returns the InstanceContext object associated with the current component instance.
- getHostName() – Returns the client host name for the client connection that is associated with this component instance.
- getPassword() – Returns the password for the client connection that is associated with this component instance.
- getPeerAddress() – Returns the client host address for the client connection that is associated with this component instance.
- getServerName() – Returns the name of the server.
- getUserName() – Returns the user name for the client connection that is associated with this component instance.
- inJaguar() – Tests if running inside the server.
- writeLog(boolean, String) – Writes a message to the server's log file.

## Jaguar.getInstanceContext()

Description

Retrieves the InstanceContext object associated with the current component instance.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

```
public InstanceContext getInstanceContext()
```

Return value

An InstanceContext object for the current component instance.

Usage

Components that do not implement the ServerBean interface can call this method to get an InstanceContext object. The InstanceContext provides transaction primitives that allow the component to influence the outcome of the transactions in which it participates.

Components that implement InstanceContext receive the InstanceContext via the ServerBean.activate(InstanceContext, String) method.

See also

InstanceContext, ServerBean

## Jaguar.getHostName()

Description

Returns the client host name for the client connection that is associated with this component instance.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

public static String getHostName() throws JException

Return value

The client host name. The host name can be 0 length if the client software did not supply the host name.

**Note**
Java clients do not supply the client host name (there is no mechanism to retrieve the host name in Java).

See also

getPeerAddress()

## Jaguar.getPassword()

Description

Returns the password for the client connection that is associated with this component instance.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

public static String getPassword() throws JException

Return value

The client password. The password can be 0 length.

Usage

getPassword returns the password for the client connection that is associated with this component instance.

This method cannot be called from a component instance that is running as a service component, since service components run without client interaction.

See also

getUserName()

## Jaguar.getPeerAddress()

Description: Returns the client host address for the client connection that is associated with this component instance.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

```
public static String getPeerAddress() throws JException
```

Return value: The client's IP address, or "0.0.0.0" if the client's IP address is unavailable.

See also: getHostName()

## Jaguar.getServerName()

Description: Returns the name of the server.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

```
public static String getServerName() throws JException
```

Return value: The name of the server.

## Jaguar.getUserName()

Description: Returns the user name for the client connection that is associated with this component instance.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

```
public static String getUserName() throws JException
```

Return value: The user name. The user name can be 0 length.

Usage: getUserName returns the user name for the client connection that is associated with this component instance.

This method cannot be called from a component instance that is running as a service component, since service components run without client interaction.

See also getPassword()

## Jaguar.inJaguar()

Description Tests if running inside the server.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

```
public static boolean inJaguar() throws JException
```

Return value true if running inside the server, false otherwise.

Usage As an alternative, you can call the method com.sybase.CORBA.ORB.isClient(), which returns a boolean value that is true if running outside of EAServer. Use this alternative if your code may be run without the EAServer server-side classes in the CLASSPATH.

## Jaguar.writeLog(boolean, String)

Description Writes a message to the server's log file.

**Standard output redirected to the server log**
Prehistoric EAServer versions required you to call this method to write to the log. In version 3.0 or later, you can call any of the System.out.print methods.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | Jaguar |

```
public static native void writeLog
   (boolean use_date, String logmsg)
   throws JException
```

Parameters *use_date*

`true` if the current date and time should be prepended to the log message; `false` otherwise.

*logmsg*

A message to be written to the server's log file.

Usage This method records a message in the server's log file.

By convention, errors that occur on the server are written to the log. Java components should call writeLog(String) rather than printing to the console with java.lang.System.out or java.lang.System.err.

For information on configuring the log file used by the server, see Chapter 3, "Creating and Configuring Servers," in the *EAServer System Administration Guide*.

# com.sybase.jaguar.server.JContext class

Description

package com.sybase.jaguar.server;

public class JContext extends Object

Instantiates objects that are used to send result sets from a Java component method and provides a method to forward rows from a java.sql.ResultSet to the client.

Constructors

None. All methods are static.

Methods

- createServerResultSetMetaData() – Creates a JServerResultSetMetaData object.
- createServerResultSet(JServerResultSetMetaData) – Creates a JServerResultSet object with row format that matches the specified JServerResultSetMetaData object.
- forwardResultSet(ResultSet) – Retrieves the rows from a java.sql.ResultSet object and forward them to the client.
- getComponentName() – Retrieves the name of the currently executing component.
- getPackageName() – Determines the name of the package in which the currently executing component is installed.

See also

JServerResultSet, JServerResultSetMetaData

## JContext.createServerResultSetMetaData()

Description

Creates a JServerResultSetMetaData object.

Syntax

| Package | com.sybase.jaguar.server |
|---|---|

| Class | JContext |
|---|---|

```
public static JServerResultSetMetaData
    createServerResultSetMetaData()
  throws SQLException
```

Usage

The JServerResultSetMetaData reference can be used to describe result rows to be sent to the client.

See also

createServerResultSet(JServerResultSetMetaData), forwardResultSet(ResultSet)

## JContext.createServerResultSet(JServerResultSetMetaData)

Description

Creates a JServerResultSet object.

Syntax

| Package | com.sybase.jaguar.server |
|---|---|
| Class | JContext |

```
public static JServerResultSet  createServerResultSet
   ( JServerResultSetMetaData metadata)
   throws SQLException
```

Parameters

*metadata*

A JServerResultSetMetaData object that has been initialized to describe the result set that will be sent.

See also

createServerResultSetMetaData(), forwardResultSet(ResultSet)

## JContext.forwardResultSet(ResultSet)

Description

Retrieves the rows from a java.sql.ResultSet object and forward them to the client.

Syntax

| Package | com.sybase.jaguar.server |
|---|---|
| Class | JContext |

```
public static void
    forwardResultSet( ResultSet rs)
  throws SQLException
```

Parameters

*rs*

A java.sql.ResultSet containing result rows from a JDBC query to a third-tier server.

See also

java.sql.ResultSet

## JContext.getComponentName()

Description

Retrieves the name of the currently executing component.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | JContext |

```
public static String
    getComponentName()
```

Return value

The name of the component.

Usage

getPackageName() and getComponentName() allow you to determine the name of the currently executing component. Within a server, components are identified by the name of the CORBA package where they are installed and the component name.

See also

getPackageName(), Jaguar.getServerName()

## JContext.getPackageName()

Description

Determines the name of the package in which the currently executing component is installed.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.server |
| Class | JContext |

```
public static String
    getPackageName()
```

Return value

The name of the CORBA package.

Usage

getPackageName() and getComponentName() allow you to determine the name of the currently executing component. Within a server, components are uniquely identified by the name of the CORBA package where they are installed and the component name.

See also getComponentName(), Jaguar.getServerName()

# com.sybase.jaguar.sql.JServerResultSet interface

Description

```
package com.sybase.jaguar.sql;
public interface JServerResultSet extends Object
```

Provides methods to send rows to the client. JServerResultSet is similar to the java.sql.ResultSet interface, which is used to retrieve result rows from a server.

Constructors Call JContext.createServerResultSet(JServerResultSetMetaData).

Methods

- done() – Indicates that all rows in a result set have been sent.
- findColumn(String) – Maps a column name to a column index.
- getMetaData() – Returns a java.sql.ResultSetMetaData object that describes the rows in a result set. The metadata includes the number of columns, the datatype of each column, and other details about each column such as whether values can be NULL.
- next() – Sends a row to the client.
- setBigDecimal(int, BigDecimal, int) – Specifies a non-NULL value for a BigDecimal column.
- setCurrency(int, long) – Specifies a non-NULL value for a column that represents a cash value.
- setNull(int) – Specifies that a column in the current row has value NULL.
- set<Object>(int, <Object>) – Specifies a non-NULL value for a column in the current row.

Usage

A JServerResultSetMetaData instance is required to construct a JServerResultSet. JServerResultSetMetaData describes the format of rows in the result set. After initializing the JServerResultSetMetaData instance, call JContext.createServerResultSet(JServerResultSetMetaData).

The cursor of a JServerResultSet is initially positioned before the first row. An initial next() call is required to move the cursor to the first row.

Subsequent calls to next() add new rows; each should be preceded by set<Object>(int, <Object>) or setNull(int) calls to set column values for the row.

You can add any number of rows with next(). Once all rows have been added, call the done() method to indicate the end of the result set.

After the done() method finishes, the JServerResultSet is again positioned before the first row. The same JServerResultSet instance can be used to another result set based on the same metadata.

Implementations of the JServerResultSet interface may buffer rows as needed during consecutive next() calls before sending them to the client. The done() method should flush any buffered rows (and flush network buffers as well, if possible—the EAServer done() implementation flushes network buffers).

See also JContext.forwardResultSet(ResultSet)

## JServerResultSet.done()

Description Indicates that all rows in a result set have been sent.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSet |

```
public abstract void done()
  throws SQLException
```

Usage You must call the done() method to indicate that all rows in a result set have been sent.

## JServerResultSet.findColumn(String)

Description Returns the index for the column that has the specified name.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSet |

```
public abstract int findColumn( String columnName)
  throws SQLException
```

Parameters *columnName*
The name of the column of interest.

Return value The index of the column whose name matches the supplied name. Throws a SQLException if no column has a matching name. The index of the first column is 1.

See also JServerResultSetMetaData.setColumnName(int, String)

## JServerResultSet.getMetaData()

Description

Returns a java.sql.ResultSetMetaData object that describes the rows in a result set. The metadata includes the number of columns, the datatype of each column, and other details about each column, such as whether values can be NULL.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSet |

```
public abstract ResultSetMetaData getMetaData()
   throws SQLException
```

Return value

A java.sql.ResultSetMetaData object that describes the rows in a result set.

Usage

A JServerResultSet object's metadata is determined when the object is constructed by calling createServerResultSetMetaData(). The metadata cannot be changed afterwards.

See also

java.sql.ResultSetMetaData, createServerResultSetMetaData(), createServerResultSet(JServerResultSetMetaData), java.sql.ResultSet.getMetaData()

## JServerResultSet.next()

Description

Sends a row to the client.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSet |

```
public abstract boolean next() throws SQLException
```

Return value

`true` if the row was successfully created, `false` otherwise.

Usage

The cursor of a JServerResultSet object is positioned before the first row when the object is constructed. An initial next() call is required to move the cursor to the first row. A done() call repositions the cursor before the first row.

After the first next() call, subsequent calls to next() add new rows; each should be preceded by set<Object>(int, <Object>) or setNull(int) calls to set column values for the row.

Any number of rows can be sent with next(). Once all rows have been sent, the done() method must be called to indicate the end of the result set.

See also done(), ResultSet.next()

## JServerResultSet.setBigDecimal(int, BigDecimal, int)

Description Specifies a non-NULL value for a java.math.BigDecimal column.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSet |

```
public abstract void setBigDecimal
   (int columnIndex,
    BigDecimal columnValue,
   int scale) throws SQLException
```

Parameters

*columnIndex*
The index of the column whose value is being set. The first column is 1.

*columnValue*
A java.math.BigDecimal value.

*scale*
The scale of the value. The scale specifies the number of decimal digits to the right of the decimal point.

Usage Use setBigDecimal methods to specify values for non-NULL java.math.BigDecimal column values. If a column's value is NULL, call setNull(int).

You can set values for columns within a row in any order.

See also ResultSet.getBigDecimal(int, int)

## JServerResultSet.setCurrency(int, long)

Description Specifies a non-NULL value for a column that represents a cash value.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSet |

```
public abstract void setCurrency
   (int columnIndex,
   long columnValue)
   throws SQLException
```

Parameters

*columnIndex*
The index of the column whose value is being set. The first column is 1.

*columnValue*
The column's value, expressed as the number of one-ten-thousandths of a cash unit. In other words, *columnValue* represents the cash value:

```
columnValue/10000
```

Usage

You must call setCurrency to specify values for columns that represent a cash value. The result set's metadata specifies whether a column represents a cash value (ResultSetMetaData.isCurrency(int) returns true for the column).

setCurrency throws a SQLException if the column does not represent a cash value.

See also

ResultSet.getBigDecimal(int, int), ResultSetMetaData.isCurrency(int), JServerResultSetMetaData.setCurrency(int, boolean)

## JServerResultSet.setNull(int)

Description

Specifies that a column in the current row has value NULL.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSet |

```
public abstract void setNull(int columnIndex)
  throws SQLException
```

Parameters

*columnIndex*
The index of the column whose value is being set. The first column is 1.

Usage

An exception is thrown if the ResultSet object's metadata does not allow NULL values for the column.

See also

JServerResultSetMetaData.setNullable(int, int), JServerResultSet.getMetaData(), ResultSet.wasNull()

## JServerResultSet.set<Object>(int, <Object>)

Description

Specifies a non-NULL value for a column in the current row.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |

Interface JServerResultSet

```
public abstract void setASCIIStream
   (int columnIndex, java.io.InputStream columnValue)
   throws SQLException, IOException

public abstract void setBinaryStream
   (int columnIndex, java.io.InputStream columnValue)
   throws SQLException, IOException

public abstract void setBoolean
   (int columnIndex, boolean columnValue)
   throws SQLException

public abstract void setByte
   (int columnIndex, byte columnValue)
   throws SQLException

public abstract void setDouble
   (int columnIndex, double columnValue)
   throws SQLException

public abstract void setDouble
   (int columnIndex, double columnValue)
   throws SQLException

public abstract void setFloat
   (int columnIndex, float columnValue)
   throws SQLException

public abstract void setInt
   (int columnIndex, int columnValue)
   throws SQLException

public abstract void setShort
   (int columnIndex, short columnValue)
   throws SQLException

public abstract void setString
   (int columnIndex, java.lang.String columnValue)
   throws SQLException

public abstract void setTimestamp
   (int columnIndex, java.sql.Timestamp columnValue)
   throws SQLException
```

Parameters

*columnIndex*

The index of the column whose value is being set. The first column is 1.

*columnValue*

An object of the appropriate type that contains the value for the column. The object type must match the column type that was specified by JServerResultSetMetaData.setColumnType(int, int) for the result set's metadata. Table 1-1 on page 40 lists type mappings.

Usage

Use the set<Object> methods to specify values for non-NULL column values. If a column's value is NULL, call setNull(int).

You can set values for columns within a row in any order.

See also

JServerResultSetMetaData.setColumnType(int, int), setBigDecimal(int, BigDecimal, int), java.sql.ResultSet

# com.sybase.jaguar.sql.JServerResultSetMetaData interface

Description

```
package com.sybase.jaguar.sql;
public interface JServerResultSetMetaData
   extends ResultSetMetaData
```

Provides methods to describe a result set's metadata. Metadata specifies the number of columns in each row as well as the datatype, format, nullability, and so forth for each column.

Constructors

The JContext.createServerResultSetMetaData() method returns a class instance that implements this interface.

Methods

- setAutoIncrement(int, boolean) – (Not yet supported.) Specifies whether a column has the auto-increment property.
- setCaseSensitive(int, String) – (Not yet supported.) Specifies whether a column's values are case-sensitive.
- setCatalogName(int, String) – (Not yet supported.) Specifies the name of the column's catalog (database).
- setColumnCount(int) – Specifies the number of columns that will be sent in result-set rows.
- setColumnDisplaySize(int, int) – Specifies the column's normal maximum width in characters.
- setColumnLabel(int, String) – Recommends a display title for the column.
- setColumnName(int, String) – Specifies the column's name.
- setColumnType(int, int) – Specifies the column's SQL (java.sql.Types) datatype.
- setColumnTypeName(int, String) – (Not yet supported.) Specifies a column's data-source-specific type name.

- setCurrency(int, boolean) – Specifies whether the column represents a cash value.
- setNullable(int, int) – Specifies whether column values can be null.
- setPrecision(int, int) – Specifies the column's precision. The precision equals the number of decimal digits in a value.
- setScale(int, int) – Specifies the column's scale. The scale equals the number of decimal digits to the right of the decimal point.
- setSchemaName(int, String) – (Not yet supported.) Specifies the schema name of the column's table.
- setSearchable(int, boolean) – (Not yet supported.) Specifies whether a column can be used in a SQL where clause.
- setSigned(int, boolean) – (Not yet supported.) Specifies whether the column represents a signed number.
- setTableName(int, String) – (Not yet supported.) Specifies the name of the table that contains the column.

**Note**
The current version does not support some interface methods. The list above indicates the methods that are not yet supported. These methods throw a JException with a "Unsupported Functionality" message.

Usage

JServerResultSetMetaData provides set methods that correspond to the get methods defined in java.sql.ResultSetMetaData. Since JServerResultSetMetaData extends ResultSetMetaData, you can call the get methods directly on a JServerResultSetMetaData object.

You can use an initialized JServerResultSetMetaData object to create one or more JServerResultSet objects by calling JContext.createServerResultSet(JServerResultSetMetaData).

See also

java.sql.ResultSetMetaData

## JServerResultSetMetaData.setColumnCount(int)

Description

Specifies the number of columns that will be sent in result-set rows.

Syntax

Package com.sybase.jaguar.sql

| Interface | JServerResultSetMetaData |
|---|---|

public abstract void setColumnCount(int columnCount)
throws SQLException

Parameters

*columnCount*
The number of columns.

Usage

You must call setColumnCount() before you can call any other methods to describe an individual column's metadata. Once the number of columns is specified, it cannot be changed without discarding any column descriptions that you have set. That is, if you call setColumnCount() again, you must reset each column's metadata.

See also

ResultSetMetaData.getColumnCount()

## JServerResultSetMetaData.setColumnDisplaySize(int, int)

Description

Specifies the column's normal maximum width in characters.

Syntax

| Package | com.sybase.jaguar.sql |
|---|---|
| Interface | JServerResultSetMetaData |

public abstract void setColumnDisplaySize
(int columnIndx, int size)
throws SQLException

Parameters

*columnIndex*
The index of the column. The first column has index 1.

*size*
The maximum width in characters.

Usage

setColumnDisplaySize determines the maximum length of variable length columns (CHAR, VARCHAR, LONGVARCHAR, BINARY, VARBINARY, LONGVARBINARY).

If you do not call setColumnDisplaySize to set a default display size, the implementation-specific default is used. To avoid excessive memory allocation, you must explicitly set the display size. In particular, the default display sizes for LONGVARCHAR and LONGVARBINARY columns can be larger than a Gigabyte.

See also

ResultSetMetaData.getColumnDisplaySize(int)

## JServerResultSetMetaData.setColumnLabel(int, String)

Description

Recommends a display title for the column.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSetMetaData |

```
public abstract void setColumnLabel
  (int columnIndex, String label)
  throws SQLException
```

Parameters

*columnIndex*

The index of the column. The first column has index 1.

*label*

The recommended display title. The default is the column name specified with setColumnName(int, String).

See also

ResultSetMetaData.getColumnLabel(int), setColumnName(int, String)

## JServerResultSetMetaData.setColumnName(int, String)

Description

Specifies the column's name.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSetMetaData |

```
public abstract void setColumnName
  (int columnIndex, String columnName)
  throws SQLException
```

Parameters

*columnIndex*

The index of the column. The first column has index 1.

*columnName*

The name of the column. The default is "" (0-length string).

See also

ResultSetMetaData.getColumnName(int)

## JServerResultSetMetaData.setColumnType(int, int)

Description

Specifies the column's SQL (java.sql.Types) datatype.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSetMetaData |

```
public abstract void setColumnType
  (int columnIndex, int SQLType)
  throws SQLException
```

Parameters

*columnIndex*

The index of the column. The first column has index 1.

*SQLType*

A symbolic constant that indicates the column's Java datatype. Constants are defined statically in the class java.sql.Types. The table below lists the supported java.sql.Types and lists, for each type, the corresponding Java type and the JServerResultSet.set<Object>(int, <Object>) method that must be called to set values for the column.

***Table 1-1: Mapping type constants to Java types and setXXX methods***

| java.sql.Types constant | Java datatype | JServerResultSet method to set values |
|---|---|---|
| BINARY | byte[] | setBinaryStream or setBytes |
| BIT | boolean | setBoolean |
| CHAR | java.lang.String | setASCIIStream or setString |
| DECIMAL | java.math.BigDecimal | setBigDecimal |
| DOUBLE | double | setDouble |
| FLOAT | double | setDouble |
| INTEGER | int | setInt |
| LONGVARBINARY | java.io.InputStream or byte[] | setBinaryStream or setBytes |
| LONGVARCHAR | String | setASCIIStream or setString |
| NUMERIC | java.math.BigDecimal | setBigDecimal |
| REAL | float | setFloat |
| SMALLINT | short | setShort |
| TIMESTAMP | java.sql.Timestamp | setTimestamp |
| TINYINT | byte | setByte |
| VARCHAR | java.lang.String | setString |
| VARBINARY | byte[] | setBytes |

**Note**
java.sql.Types.OTHER and java.sql.Types.BIGINT are not supported.

Usage

setColumnType(int, int) specifies the datatype for a column. There is no default. For java.math.BigDecimal columns, you must also call setPrecision(int, int) and setScale(int, int) to specify the column's precision and scale, respectively.

For columns that represent cash values, you must use JServerResultSet.setCurrency(int, long) to set values for the column.

See also

java.sql.Types, ResultSetMetaData.getColumnType(int), setPrecision(int, int), setScale(int, int)

## JServerResultSetMetaData.setCurrency(int, boolean)

Description

Specifies whether the column represents a cash value.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSetMetaData |

```
public abstract void
 setCurrency
   (int columnIndex, boolean property)
   throws SQLException
```

Parameters

*columnIndex*

The index of the column. The first column has index 1.

*property*

`true` if the column represents a cash value, `false` otherwise. The default is `false`.

See also

ResultSetMetaData.isCurrency(int)

## JServerResultSetMetaData.setNullable(int, int)

Description

Specifies whether column values can be null.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSetMetaData |

```
public abstract void setNullable
   (int columnIndex, int property)
   throws SQLException
```

Parameters

*columnIndex*

The index of the column. The first column has index 1.

*property*

A symbolic constant that takes the following values:

| Value | To indicate |
|---|---|
| columnNullable | Values for the column can be null. |
| columnNoNulls | Values for the column cannot be null. |
| columnNullableUnknown | Nullability of the column is not known. |

The default is columnNullableUnknown.

See also

JServerResultSet.setNull(int), ResultSetMetaData.isNullable(int)

## JServerResultSetMetaData.setPrecision(int, int)

Description

Specifies the column's precision. The precision equals the number of decimal digits in a value.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSetMetaData |

```
public abstract void setPrecision
  (int columnIndex, int precision)
  throws SQLException
```

Parameters

*columnIndex*

The index of the column. The first column has index 1.

*precision*

The precision of the column. The default is 0.

Usage

This method applies to java.math.BigDecimal columns only.

See also

ResultSetMetaData.getPrecision(int), setScale(int, int)

## JServerResultSetMetaData.setScale(int, int)

Description

Specifies the column's scale. The scale equals the number of decimal digits to the right of the decimal point.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.sql |
| Interface | JServerResultSetMetaData |

```
public abstract void setScale
  (int columnIndex, int scale)
  throws SQLException
```

Parameters

*columnIndex*

The index of the column. The first column has index 1.

*scale*

The scale for the column. The default is 0.

Usage

This method applies to java.math.BigDecimal columns only.

See also

ResultSetMetaData.getScale(int), setPrecision(int, int)

# com.sybase.jaguar.util.JException class

Description

```
package com.sybase.jaguar.util;
public class JException
  extends Exception
```

JException is the generic exception that is thrown by methods in the EAServer classes or in generated client stub classes.

Constructors — Same as java.lang.Exception.

Methods — Same as java.lang.Exception.

See also — JConnectionNotFoundException, java.sql.SQLException

CHAPTER 2

# C Routines Reference

This chapter contains reference pages for the C routines that are provided for use by EAServer C or C++ components. Routines are indexed in the following sections:



Detailed reference pages for each routine follow the index sections. Routines are listed in alphabetical order by routine name.

## Alphabetical list of all routines

- JagAlloc – Allocate memory for use in C component code.
- JagCmGetCachebyName – Retrieve the handle for the data source with the specified name.
- JagCmGetCachebyUser – Retrieve a data source handle for connections that use a specified set of values for server, user name, password, and connectivity library.
- JagCmGetConnection – Retrieve a connection from a specified data source or from any data source that matches a specified set of values for server, user name, password, and connectivity library.
- JagCmGetCtx – Obtain the connectivity-library-specific context reference that is used to allocate connections from a data source.

- JagCmGetProxyConnection – Retrieve a cached connection, specifying an alternate login name to set-proxy to.
- JagCmReleaseConnection – Place a connection back in the data source for reuse.
- JagCompleteWork – Indicate that the component's work for the current transaction was successfully finished and that this component instance should be deactivated.
- JagContinueWork – State indicator routine to specify that the component's work for the current transaction may be committed.
- JagDisallowCommit – State indicator routine to specify that the current transaction cannot be committed because the component's work has not been completed.
- JagFree – Free memory that was allocated with JagAlloc.
- JagGetHostName – Retrieve the client host name for the client connection that is associated with a C or C++ component instance.
- JagGetPassword – Retrieve the password for the client connection that is associated with a C or C++ instance.
- JagGetPeerAddress – Retrieve the client host IP address for the client connection that is associated with a C or C++ component instance.
- JagGetUserName – Retrieve the user name for the client connection that is associated with a C or C++ component instance.
- JagInTransaction – Determine whether the current method is executing in a transaction.
- JagIsRollbackOnly – Query whether the current transaction is doomed to be rolled back or is still viable.
- JagLog – Write a message to the server's log file.
- JagRollbackWork – Indicate that the component cannot complete its work for the current transaction. The component instance will be deactivated when the method returns.
- JagSleep – Suspend execution of the thread in which your component is running.

## Routines for managing transaction flow

A component that participates in transactions can call these routines to influence the outcome of the current transaction.

- JagCompleteWork – Indicate that the component's work for the current transaction was successfully finished and that this component instance should be deactivated when the method returns.
- JagContinueWork – Indicate that the component should not be deactivated after the current method invocation; allow the current transaction to be committed if the component instance is deactivated.
- JagDisallowCommit – Indicate that the current transaction cannot be committed because the component's work has not been completed; the instance remains active after the current method returns.
- JagInTransaction – Determine whether the current method is executing in a transaction.
- JagIsRollbackOnly – Query whether the current transaction is doomed to be rolled back or is still viable.
- JagRollbackWork – Indicate that the component cannot complete its work for the current transaction. The component instance will be deactivated when the method returns.

## Routines for managing cached connections

EAServer provides the following routines to manage cached connections:

- JagCmGetCachebyName – Retrieve the handle for the data source with the specified name.
- JagCmGetCachebyUser – Retrieve a data source handle for connections that use a specified set of values for server, user name, password, and connectivity library.
- JagCmGetConnection – Retrieve a connection from a specified data source or from any data source that matches a specified set of values for server, user name, password, and connectivity library.
- JagCmGetCtx – Obtain the connectivity-library-specific context reference that is used to allocate connections from a data source.
- JagCmGetProxyConnection – Retrieve a cached connection, specifying an alternate login name to set-proxy to.

- JagCmReleaseConnection – Place a connection back in the data source for reuse.

## Routines for handling errors in C or C++ components

These routines are useful for handling errors in C components.

- JagLog – Write a message to the server's log file.

## Routines for managing memory in C or C++ components

- JagAlloc – Allocate memory for use in C component code.
- JagFree - Free memory that was allocated with JagAlloc.

## Routines to obtain user login information

You can call these routines in C or C++ component code to obtain information about the client connection that is associated with the current instance:

- JagGetHostName – Retrieve the client host name for the client connection that is associated with a C or C++ component instance.
- JagGetPassword – Retrieve the password for the client connection that is associated with a C or C++ component instance.
- JagGetPeerAddress – Retrieve the client host IP address for the client connection that is associated with a C or C++ component instance.
- JagGetUserName – Retrieve the user name for the client connection that is associated with a C or C++ component instance.

# Unsupported routines

These routines are no longer supported in EAServer 6.0 and later releases:

- JagBeginResults
- JagBindCol

- JagCmCacheProps
- JagCmGetCtx
- JagColAttributes
- JagDescribeCol
- JagEndResults
- JagFreeCollectionHandle
- JagFreeCollectionList
- JagFreeSharedDataHandle
- JagGetCollection
- JagGetCollectionList
- JagGetInstanceData
- JagGetSharedData
- JagGetSharedDataByIndex
- JagGetSharedValue
- JagLockCollection
- JagLockNoWaitCollection
- JagNewCollection
- JagNewSharedData
- JagNewSharedDataByIndex
- JagResultsPassthrough
- JagSendMsg
- JagSetSharedValue
- JagSetInstanceData
- JagUnlockCollection

# JagAlloc

Description Allocate memory for use in C component code.

Syntax

```
void * JAG_PUBLIC JagAlloc(
    SQLINTEGER len
    );
```

Parameters

*len*
The number of bytes to be allocated.

Return value

A pointer to newly allocated memory or NULL if the requested block of memory can not be allocated.

Usage

In C components, memory used to store output parameters for variable-length types (string and binary) must be allocated with JagAlloc.

Memory allocated with JagAlloc must be freed with JagFree.

In C++ components, use the standard CORBA memory allocation and deallocation routines.

See also

JagFree

# JagCmGetCachebyName

Description

Retrieve the handle for the data source with the specified name.

**Note** Beginning in EAServer 6.0, all data sources allow access by name.

Syntax

```
JagStatus JagCmGetCachebyName (
    SQLCHAR      *cachename,
    JagCmCache   *cache
    );
```

Parameters

*cachename*
The data source name.

*cache*
The address of a JagCmCache handle. If a matching data source is available, its handle is returned as **cache*. If no matching data source exists, **cache* is set to NULL.

Return value

| Return value | To indicate |
|---|---|
| JAG_SUCCEED | Success. **cache* is set to the address of the matching data source. |
| JAG_FAIL | Failure. |

JagCmGetCachebyName fails for the following reasons:

- A NULL value was passed for *cachename*.
- No matching data source was found.
- A matching data source is installed, but the data source properties do not allow retrieval with JagCmGetCachebyName.

JagCmGetCachebyName records a message that describes the failure reason in the server log file.

Usage

JagCmGetCachebyName allows you to retrieve connections without specifying the user name, password, and other parameters that are required by the JagCmGetCachebyUser routine.

You can retrieve a data source handle with either JagCmGetCachebyUser or JagCmGetCachebyName. Calling JagCmGetCachebyName allows you to change the data source user name, password, or server properties without requiring corresponding changes to your component source code.

See also

JagCmGetCachebyUser

# JagCmGetCachebyUser

Description

Retrieve a data source handle for connections that use a specified set of values for server, user name, password, and connectivity library.

Syntax

```
JagStatus JagCmGetCachebyUser (
    SQLCHAR     *username,
    SQLCHAR     *password,
    SQLCHAR     *server,
    SQLCHAR     *con_lib,
    JagCmCache  *cache
    );
```

Parameters

*username*

The user name for connections in the desired data source.

*password*

The password used by connections in the desired data source.

*server*

For ODBC connections, the ODBC data source name (as you would use to call SQLConnect). For Client-Library connections, the server name (as you would use to call ct_connect).

*con_lib*

A string value indicating the connectivity library used by connections in the data source. Allowable values are:

| con_lib value | To indicate |
|---|---|
| "CTLIB_110" | Sybase Open Client Client-Library |
| "ODBC" | An ODBC implementation library |
| "OCI_7" | Oracle Call Interface 7.x |
| "OCI_8" | Oracle Call Interface 8.x |

*cache*

The address of a JagCmCache handle. If a matching data source is available, its handle is returned as **cache*. If no matching data source exists, **cache* is set to NULL.

Return value

| Return value | To indicate |
|---|---|
| JAG_SUCCEED | Success. **cache* is set to the address of the matching data source. |
| JAG_FAIL | Failure. |

JagCmGetCachebyUser fails for the following reasons:

- A NULL value was passed for *username*, *password*, *server*, or *con_lib*.
- An invalid value was passed for *con_lib*.
- No matching data source was found.

Usage

JagCmGetCachebyUser allows you to retrieve connections that match the desired characteristic values for:

- Server name
- User name
- Password
- Connectivity library

You can use this routine when you are not sure if a data source is configured for a particular set of characteristic values. If no such data source is available, JagCmGetCachebyUser sets the **cache* parameter to NULL. If one or more matching data sources exist, JagCmGetCachebyUser sets **cache* to the handle for the first matching data source that it finds.

See JagCmGetConnection for an example that calls JagCmGetCachebyUser.

See also JagCmGetCachebyName

# JagCmGetConnection

Description Retrieve a connection from a specified data source or from any data source that matches a specified set of values for server, user name, password, and connectivity library.

Syntax

```
JagStatus JagCmGetConnection (
    JagCmCache    *cache,
    SQLCHAR       *username,
    SQLCHAR       *password,
    SQLCHAR       *server,
    SQLCHAR       *con_lib,
    SQLPOINTER    *connection,
    JagCmOpt      opt
    );
```

Parameters

*cache*

The address of a JagCmCache cache handle variable. The input value determines how the parameter is used:

- If **cache* is not NULL, it must specify a valid data source handle. JagCmGetConnection attempts to return a connection from the specified data source. You can call JagCmGetCachebyUser to obtain a data source handle for any data source.
- If **cache* is NULL, characteristic values for *username*, *password*, *server*, and *con_lib* must be supplied. If a matching data source is found, **cache* is set to handle for the data source.

*username*

When **cache* is NULL, the user name for connections in the desired data source. Ignored when **cache* is not NULL.

*password*

When **cache* is NULL, the password used by connections in the desired data source. Ignored when **cache* is not NULL.

*server*

When **cache* is NULL, the name of the server to which cached connections are made. Ignored when **cache* is not NULL.

*con_lib*

When **cache* is NULL, indicates a string value indicating the connectivity library used by connections in the data source. Ignored when **cache* is not NULL.

When **cache* is NULL, allowable values for *con_lib* are:

| con_lib value | To indicate |
|---|---|
| "CTLIB_110" | Sybase Open Client Client-Library |
| "ODBC" | An ODBC implementation library |
| "OCI_7" | Oracle Call Interface 7.x |
| "OCI_8" | Oracle Call Interface 8.x |

*connection*

The address of a variable that receives the connection handle. Declare a variable of the appropriate type, as follows:

- For ODBC connections, pass the address of an SQLHDBC variable
- For Client-Library connections, pass the address of a CS_CONNECTION * variable
- For Oracle 7.x connections, pass the address of an OCI Lda_Def variable
- For Oracle 8.x connections, pass the address of an OCI OCISvcCtx variable

On successful return, the connection will be open and in a state that allows commands to be sent to the remote server.

*opt*

A symbolic value that indicates the desired behavior if all connections in a data source are in use. Allowable values are:

| Value of opt | JagCmGetConnection behavior when all connections are in use |
|---|---|
| JAG_CM_NOWAIT | Fails with an error if no connection can be returned. |
| JAG_CM_WAIT | Does not return until a connection becomes available. |
| JAG_CM_FORCE | Allocates and opens a new connection. The new connection is not cached and will be destroyed when JagCmReleaseConnection is called. |

Return value

| Return value | To indicate |
|---|---|
| ODBC status code | The result of a SQLAllocConnect or SQLConnect call, or SQL_SUCCESS in the case where a previously opened connection is returned. |
| Client-Library status code | The result of a ct_con_alloc or ct_connect call, or CS_SUCCEED in the case where a previously opened connection is returned. |
| OCI_SUCCESS (An OCI 7.x and 8.x status code) | Successful retrieval of an OCI 7.x or 8.x connection. |
| OCI_FAIL (An OCI 7.x and 8.x status code) | Failure to retrieve an OCI 7.x or 8.x connection. Check the server log for errors, and verify that the connection can be pinged. |
| JAG_FAIL | Failure. JagCmGetConnection returns JAG_FAIL when the call specifies an invalid *con_lib* value. |

Usage

JagCmGetConnection returns a connection that was allocated and opened with the specified connectivity library and that has matching values for server, user name, and password.

JagCmGetConnection behaves differently depending on whether the **cache* parameter is NULL.

Calls that pass a NULL data source handle

If **cache* is NULL, CmGetConnection looks for a data source with settings that match the values of the *username*, *password*, *server*, and *con_lib* parameters. If a cache is found and a connection is available, a connection is returned from that data source and **cache* is set to reflect the data source from which the connection came. If no data source is found, then a connection structure is allocated, a connection is opened using the specified connectivity library and the new connection structure is returned. If a data source was found, *con_lib* is ignored. The following table summarizes the JagCmGetConnection call when **cache* is NULL.

***Table 2-1: JagCmGetConnection behavior when *cache is NULL***

| Data source found? | Connection available in data source? | Result |
|---|---|---|
| Yes | Yes | The call returns a connection handle in **connection* and sets **cache* to reflect the data source from which the connection came. |

| Data source found? | Connection available in data source? | Result |
|---|---|---|
| Yes | No | Depending on the value of the *opt* parameter, the call fails, waits for an available connection, or allocates and opens a new, uncached connection. **cache* is returned as NULL. |
| No | N/A | The call attempts to allocate and open a new, uncached connection. **cache* is returned as NULL. |

### Cached and uncached connections

A connection obtained with JagCmGetConnection is either cached or uncached.

A *cached connection* is one that was taken from a configured data source. When JagCmGetConnection returns a cached connection, it sets **cache* to indicate the data source to which the connection belongs. Cached connections must be released to the data source from which they were taken: pass the data source reference obtained in the JagCmGetConnection call when calling JagCmReleaseConnection.

An *uncached connection* is one that was not taken from a data source. JagCmGetConnection returns an uncached connection in either of the following cases:

- There is no data source configured with the specified *username*/*password*/*server*/*con_lib* parameter values.
- There is a matching data source, all its connections are in use, and the JagCmGetConnection call specifies JAG_CM_FORCE as the value of the *opt* parameter.

### Calls that pass a non-NULL data source handle

When a data source handle is passed in **cache*, JagCmGetConnection looks for an available connection in that data source. If none is available, then the value of the *opt* parameter determines whether the call waits for a connection to be released, fails, or opens a new, uncached connection.

See also JagCmReleaseConnection

# JagCmGetCtx

Description

Obtain the connectivity-library-specific context reference that is used to allocate connections from a data source.

Syntax

```
JagStatus JagCmGetCtx (
    JagCmCache    *cache,
    SQLCHAR       *username,
    SQLCHAR       *password,
    SQLCHAR       *server,
    SQLCHAR       *con_lib,
    SQLPOINTER    *ctx
    );
```

Parameters

*cache*

The address of a JagCmCache data source handle variable. The input value determines how the parameter is used:

- When **cache* is NULL, the values of *username*, *password*, *server*, and *con_lib* are used to search for a matching data source. If found, **ctx* is set to the address of the connectivity-library context handle, and **cache* is set to the matching data source handle.
- If **cache* contains a valid data source handle, JagCmGetCtx retrieves the connectivity-library context for the indicated data source. You can call JagCmGetCachebyUser or JagCmGetCachebyName to obtain a data source handle for any data source.

*username*

When **cache* is NULL, the user name for connections in the desired data source. Ignored when **cache* is not NULL.

*password*

When **cache* is NULL, the password used by connections in the desired data source. Ignored when **cache* is not NULL.

*server*

When **cache* is NULL, the name of the server to which cached connections are made. Ignored when **cache* is not NULL.

*con_lib*

When **cache* is NULL, a string value indicating the connectivity library used by connections in the data source. Ignored when *cache* is not NULL.

When *cache* is NULL, *con_lib* must be one of the following:

| con_lib value | To indicate |
|---|---|
| "CTLIB_110" | Sybase Open Client Client-Library |
| "ODBC" | An ODBC implementation library |

*ctx*

The address of a variable that receives the connectivity library context used to allocate cached connections. The returned type depends on the connectivity library, as follows:

| Connectivity library | Value returned in *ctx |
|---|---|
| Client-Library | A pointer to a CS_CONTEXT structure. Each data source uses a separate CS_CONTEXT structure. |
| ODBC | An ODBC SQLHENV environment handle. This handle is shared by all ODBC data sources. |

Return value

| Returns | To indicate |
|---|---|
| JAG_SUCCEED | Successful retrieval of the CS_CONTEXT for a Client-Library data source. |
| JAG_FAIL | Failure. JagCmGetCtx fails when *con_lib* specifies an invalid value. |

JagCmGetCtx fails for the following reasons:

- The *cache* parameter is passed as NULL.
- The value of *cache* is not NULL, and **cache* references an invalid data source.
- The value of *cache* is NULL, and there is no data source matching the values specified for the *username*, *password*, *server*, and *con_lib* parameters.

Usage

JagCmGetCtx retrieves the context or environment handle that is used to allocate connections in a data source.

See also

JagCmGetConnection

# JagCmGetProxyConnection

Description

Retrieve a cached connection, specifying an alternate login name to set-proxy to.

---

**Not all data sources support set-proxy**
JagCmGetProxyConnection cannot be used with OCI connections. You must be connected to a database server, such as Adaptive Server Enterprise 11.5, that supports the set session authorization command. Set-proxy support must be enabled in the data source properties before you can use this feature. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

---

Syntax

```
JagStatus JAG_PUBLIC JagCmGetProxyConnection (
        JagCmCache     *cache,
        SQLCHAR        *username,
        SQLCHAR        *password,
        SQLCHAR        *server,
        SQLCHAR        *con_lib,
        SQLPOINTER     *connection,
        JagCmOpt       opt,
        SQLCHAR        *proxy
        );
```

Parameters

*cache*

The address of a JagCmCache data source handle variable. The input value determines how the parameter is used:

- When **cache* is NULL, the values of *username*, *password*, *server*, and *con_lib* are used to search for a matching data source. If found, **ctx* is set to the address of the connectivity-library context handle, and **cache* is set to the matching data source handle.
- If **cache* contains a valid data source handle, JagCmGetProxyConnection retrieves the connectivity-library context for the indicated data source. You can call JagCmGetCachebyUser or JagCmGetCachebyName to obtain a data source handle for any data source.

*username*

When **cache* is NULL, the user name for connections in the desired data source. Ignored when **cache* is not NULL.

*password*
When **cache* is NULL, the password used by connections in the desired data source. Ignored when **cache* is not NULL.

*server*
When **cache* is NULL, the name of the server to which cached connections are made. Ignored when **cache* is not NULL.

*con_lib*
When **cache* is NULL, a string value indicating the connectivity library used by connections in the data source. Ignored when *cache* is not NULL.

When *cache* is NULL, *con_lib* must be one of the following:

| con_lib value | To indicate |
|---|---|
| "CTLIB_110" | Sybase Open Client Client-Library |
| "ODBC" | An ODBC implementation library |

*connection*
The address of a variable that receives the connection handle. Declare a variable of the appropriate type, as follows:

- For ODBC connections, pass the address of an SQLHDBC variable
- For Client-Library connections, pass the address of a CS_CONNECTION * variable

On successful return, the connection will be open and in a state that allows commands to be sent to the remote server.

*opt*
A symbolic value that indicates the desired behavior if all connections in a data source are in use. Allowable values are:

| Value of opt | JagCmGetConnection behavior when all connections are in use |
|---|---|
| JAG_CM_NOWAIT | Fails with an error if no connection can be returned. |
| JAG_CM_WAIT | Does not return until a connection becomes available. |
| JAG_CM_FORCE | Allocates and opens a new connection. The new connection is not cached and will be destroyed when JagCmReleaseConnection is called. |

*proxy*
The user name to set-proxy to.

Return value

| Return value | To indicate |
|---|---|
| ODBC status code | The result of a SQLAllocConnect or SQLConnect call, or the set session authorization command. |
| Client-Library status code | The result of a ct_con_alloc or ct_connect call, or the set session authorization command. |
| JAG_FAIL | Failure. JagCmGetConnection returns JAG_FAIL when the call specifies an invalid *con_lib* value. |

Usage

JagCmGetProxyConnection retrieves a cached connection, specifying an alternate login name to set-proxy to. Set-proxy support must be enabled in the data source properties. If support is enabled, connections retrieved from the data source with JagCmGetConnection set-proxy to the client user name. Call JagCmGetProxyConnection to specify a different user name to set-proxy to.

Other than the set-proxy behavior, JagCmGetProxyConnection is identical to JagCmGetConnection.

See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for information on defining data sources and enabling set-proxy support.

See also

JagCmGetConnection

# JagCmReleaseConnection

Description

Place a connection back in the data source for reuse.

Syntax

```
JagStatus JagCmReleaseConnection (
    JagCmCache    *cache,
    SQLCHAR       *username,
    SQLCHAR       *password,
    SQLCHAR       *server,
    SQLCHAR       *con_lib,
    SQLPOINTER    connection,
    SQLINTEGER    opt
    );
```

Parameters

*cache*

The address of a JagCmCache data source handle variable. **cache* can be NULL or a valid data source handle.

If **cache* is not NULL, must be the data source handle that was used to obtain the connection by calling JagCmGetConnection.

If **cache* is NULL, JagCmReleaseConnection attempts to place the connection in a data source that has available space and that uses the same values for *username*, *password*, *server*, and *con_lib*. If no such data source has available space, the connection is closed and deallocated.

*username*

The user name of the connection. Ignored unless *cache* is NULL.

*password*

The password used by the connection. Ignored unless *cache* is NULL.

*server*

The name of the server to which the connection is made. Ignored unless *cache* is NULL.

*con_lib*

A string value indicating the connectivity library used by the connection. Ignored unless *cache* is NULL. Allowable values for *con_lib* are:

| con_lib value | To indicate |
|---|---|
| “CTLIB_110” | Sybase Open Client Client-Library |
| “ODBC” | An ODBC driver library |
| “OCI_7” | Oracle Call Interface 7.x |
| “OCI_8” | Oracle Call Interface 8.x |

*connection*

The connection handle to be released. The connection must be in a state that allows commands to be sent to the remote server. If commands were sent using the connection, the results of the commands must have been completely processed.

*opt*

One of the following symbolic constants:

| opt value | To indicate |
|---|---|
| JAG_CM_DROP | The connection should be forced closed and deallocated. If the connection came from a data source, a new connection will be created in its place. |
| JAG_CM_UNUSED | Normal behavior: a connection taken from a data source is placed back in the data source; a connection created outside of a data source is closed and destroyed. |

Use JAG_CM_DROP to destroy a connection when errors have made it unusable.

Return value

| Returns | To indicate |
|---|---|
| ODBC or Client-Library return status | The result of connectivity library calls to close and deallocate a connection that was not released to a data source. |
| CS_SUCCEED | A Client-Library connection was returned to a data source. |
| SQL_SUCCESS | An ODBC connection was returned to a data source. |
| JAG_FAIL | Failure. JagCmReleaseConnection fails when *cache* is NULL and *con_lib* specifies an invalid value. |

Usage

JagCmReleaseConnection releases control of a connection that was obtained from JagCmGetConnection.

**Warning!** Do not release a connection more than once.

See also

JagCmGetConnection

# JagCompleteWork

Description

Indicate that the component's work for the current transaction has been successfully completed and is ready to be committed.

Syntax

```
void JagCompleteWork();
```

Usage

JagCompleteWork specifies that the component has successfully completed its contribution to the current transaction. The component instance deactivates when control returns from the current component method invocation.

If the component instance is the initiator of the transaction (that is, it was instantiated directly by a base client), then the component dispatcher attempts to commit the transaction. The transaction commits unless the commit is disallowed or vetoed; depending on the components that are participating, this can happen in any of the following ways:

- A participating C or C++ component has called JagDisallowCommit.
- A participating Java component throws an exception from its ServerBean.deactivate() method.
- A participating ActiveX component has called IObjectContext::disableCommit().

If a component is not transactional, then JagCompletework and JagRollbackWork have the same effect: both cause the component instance to deactivate after the currently executing method returns.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also

JagContinueWork, JagDisallowCommit, JagRollbackWork

# JagContinueWork

Description

Indicate that the component should not be deactivated after the current method invocation; allow the current transaction to be committed if the component instance is deactivated.

Syntax

void JagContinueWork();

Usage

JagContinueWork specifies that the component instance should not be automatically deactivated after the current method completes. If the instance is deactivated before the next method invocation, the current transaction is committed.

When a method calls JagContinueWork, the component instance is not deactivated until one of the following happens:

- The component's stub is destroyed explicitly by the client.

- The client disconnects without explicitly destroying the stub (the current transaction is always rolled back in this case).
- The component instance calls JagCompleteWork or JagRollbackWork during a subsequent method invocation.

JagContinueWork and JagDisallowCommit allow components to maintain state between method calls. If a component is not transactional, JagContinueWork and JagDisallowCommit have the same effect: both prevent immediate deactivation of the component.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also JagCompleteWork, JagDisallowCommit, JagRollbackWork

# JagDisallowCommit

Description Indicate that the current transaction cannot be committed because the component's work has not been completed; the instance remains active after the current method returns.

Syntax
```
void JagDisallowCommit();
```

Usage JagDisallowCommit specifies that the component instance should not be automatically deactivated after the current method completes. If the instance is deactivated before the next method invocation, the current transaction is rolled back.

When a method calls JagDisallowCommit, the component instance is not deactivated until one of the following happens:

- The component's stub is destroyed explicitly by the client.
- The client disconnects without explicitly destroying the stub (the current transaction is always rolled back in this case).
- The component instance calls JagCompleteWork or JagRollbackWork during a subsequent method invocation.

JagContinueWork and JagDisallowCommit allow components to maintain state between method calls. If a component is not transactional, JagContinueWork and JagDisableCommit have the same effect: both prevent immediate deactivation of the component.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also JagCompleteWork, JagContinueWork, JagIsRollbackOnly, JagRollbackWork

# JagFree

Description Free memory that was allocated with JagAlloc.

Syntax
```
void JAG_PUBLIC JagFree(
    void *ptr
    );
```

Parameters

*ptr*

A pointer to the memory to be freed.

See also JagAlloc

# JagGetHostName

Description Retrieve the client host name for the client connection that is associated with a C or C++ component instance.

Syntax
```
JagStatus JAG_PUBLIC JagGetHostName(
      SQLPOINTER hostName,
      SQLINTEGER hostNameLen,
      SQLINTEGER *returnLen)
```

Parameters

*hostName*

The address of a character array to receive the client host name or, if the client software did not supply a host name, a zero-length string.

---

**Java clients and JagGetHostName**
Java clients do not supply the client host name (there is no mechanism to retrieve the host name in Java).

---

*hostNameLen*

The length, in bytes, of the *hostName* array. The length must include space for a null-terminator.

*returnLen*
NULL or the address of a SQLINTEGER variable.

*returnLen* is an optional output parameter that receives the length, in bytes, of the *hostName* value. The host name is null-terminated and the length includes the null-terminator.

Return value

| Return value | To indicate |
|---|---|
| JAG_SUCCEED | Success |
| JAG_FAIL | Failure |

JagGetHostName fails for the following reasons:

- *hostName* was NULL.
- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetHostName fails.

See also JagGetPeerAddress

# JagGetPassword

Description Retrieve the password for the client connection that is associated with a C or C++ component instance.

Syntax

```
JagStatus JAG_PUBLIC JagGetPassword(
        SQLPOINTER password,
        SQLINTEGER passwordLen,
        SQLINTEGER *returnLen)
```

Parameters

*password*
The address of a character array to receive the client password. If the connection has a NULL password, JagGetPassword writes a null-terminator to the *password* buffer.

*passwordLen*
The length, in bytes, of the *password* array. The length must include space for a null-terminator.

*returnLen*
NULL or the address of a SQLINTEGER variable.

*returnLen* is an optional output parameter that receives the length, in bytes, of the *password* value. The host name is null-terminated and the length includes the null-terminator.

Return value

| Return value | To indicate |
|---|---|
| JAG_SUCCEED | Success |
| JAG_FAIL | Failure |

JagGetPassword fails for the following reasons:

- *password* was NULL.
- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetPassword fails.

See also JagGetHostName, JagGetUserName

# JagGetPeerAddress

Description Retrieve the client host IP address for the client connection that is associated with a C or C++ component instance.

Syntax
```
JagStatus JAG_PUBLIC JagGetPeerAddress(
        SQLPOINTER peerAddress,
        SQLINTEGER bufLen,
        SQLINTEGER *returnLen)
```

Parameters

*peerAddress*
The address of a character array to receive the client IP address. The output value is "0.0.0.0" if the client's IP address is unavailable.

*bufLen*
The length, in bytes, of the *peerAddress* array. The length must include space for a null-terminator.

*returnLen*

NULL or the address of a SQLINTEGER variable.

*returnLen* is an optional output parameter that receives the length, in bytes, of the *peerAddress* value. The host name is null-terminated and the length includes the null-terminator.

Return value

| Return value | To indicate |
|---|---|
| JAG_SUCCEED | Success |
| JAG_FAIL | Failure |

JagGetPeerAddress fails for the following reasons:

- *peerAddress* was NULL.
- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetPeerAddress fails.

See also

JagGetHostName

# JagGetUserName

Description

Retrieve the user name for the client connection that is associated with a C or C++ component instance.

Syntax

```
JagStatus JAG_PUBLIC JagGetUserName(
        SQLPOINTER userName,
        SQLINTEGER userNameLen,
        SQLINTEGER *returnLen)
```

Parameters

*userName*

The address of a character array to receive the user name. The user name can have 0 length if no user name was supplied. In this case, only a null-terminator will be written to **userName*. (In practice, a user name is required to connect to the server unless user authentication is disabled.)

*userNameLen*

The length, in bytes, of the *userName* array. The length must include space for a null-terminator.

*returnLen*
NULL or the address of a SQLINTEGER variable.

*returnLen* is an optional output parameter that receives the length in bytes of the *userName* value. The user name is null-terminated and the length includes the null-terminator.

Return value

| Return value | To indicate |
|---|---|
| JAG_SUCCEED | Success. |
| JAG_FAIL | Failure. |

JagGetUserName fails for the following reasons:

- *userName* was NULL.
- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetUserName fails.

See also JagGetHostName, JagGetPassword

# JagInTransaction

Description Determine whether the current method is executing in a transaction.

Syntax JagBoolean JagInTransaction();

Usage Methods can call JagInTransaction to determine whether they are executing within a transaction. Methods in components that are declared to be transactional always execute as part of a transaction.

See also JagIsRollbackOnly

# JagIsRollbackOnly

Description Query whether the current transaction is doomed to be rolled back or is still viable.

Syntax

JagBoolean JagIsRollbackOnly()

Return value

JAG_TRUE if the current transaction is doomed, in other words, it can never be committed. If executing outside of any transaction, returns JAG_FALSE.

Usage

Transactional components that issue intercomponent method calls should call JagIsRollbackOnly afterward to determine whether the current transaction is still viable. If not, the method should clean up and call JagRollbackWork to deactivate the current instance.

Transactions are doomed when a participating component has called JagRollbackWork (or its equivalent if the component is a Java or ActiveX component). Work performed by participating components is rolled back when the root component of the transaction deactivates.

See also

JagInTransaction, JagRollbackWork

# JagLog

Description

Write a message to the server's log file.

Syntax

```
#include <jagpublic.h>

JagStatus JagLog(
        JagBoolean use_date,
        SQLPOINTER logmsg)
```

Parameters

*use_date*

Pass as JAG_TRUE to indicate that the message should be preceded by a timestamp in the log; pass as JAG_FALSE to log the message without a timestamp.

*logmsg*

A null-terminated string containing the message to be logged. The message must include a newline at the end.

Return value

| Return value | To indicate |
|---|---|
| JAG_SUCCEED | Success. |
| JAG_FAIL | Failure. JagLog fails if the log file can not be opened or if *logmsg* is NULL. If the log file cannot be opened, log messages are written to the server process' standard error device. |

# JagRollbackWork

Description
Indicate that the component cannot complete its work for the current transaction. The component instance will be deactivated when the method returns.

Syntax
```
void JagRollbackWork();
```

Usage
JagRollbackWork specifies that the component cannot complete its work for the current transaction. The transaction will be rolled back when the initiating component is deactivated.

If a component is not transactional, then JagRollbackWork and JagRollbackWork have the same effect: both cause the component instance to deactivate after the currently executing method returns.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also
JagCompleteWork, JagContinueWork, JagDisallowCommit, JagInTransaction, JagIsRollbackOnly

# JagSleep

Description
Suspend execution of the thread in which your component is running.

Syntax
```
void JAG_PUBLIC JagSleep (
        JagLong seconds)
```

Parameters
*seconds*
The number of seconds to sleep.

Usage
JagSleep suspends execution of the thread in which the current component instance is running. JagSleep is useful in service components that perform background processing in the run method. run typically loops forever, and calling JagSleep prevents your component from dominating the server's CPU execution time.

JagSleep can only be called by a component that is executing within EAServer. This routine is not available to clients.

**Warning!** In EAServer components, never call the sleep system routine or any other routine that suspends execution of the current process. Doing so suspends execution of the server. JagSleep suspends only the current thread, allowing components running in other threads to continue execution.

APPENDIX A

# Deprecated Java Classes and Interfaces

This appendix documents obsolete EAServer Java classes and interfaces, which are based on an obsolete version (version 0.4) of the Enterprise Java Beans specification.

Rather than using these models for developing new Java components, use the latest EJB version, for portability to other J2EE based application servers.

## Package Index

### com.sybase.jaguar.beans.enterprise

Classes and interfaces used to implement Java components and to create stubs for remote communication. These classes are based on an early draft of the Enterprise JavaBeans specification. Future releases of the Java Developer's Kit will likely provide built-in classes with the same functionality:

- com.sybase.jaguar.beans.enterprise.EnterpriseBeanException class – Exception that can be thrown by components that implement the ServerBean interface..
- com.sybase.jaguar.beans.enterprise.InstanceContext interface – An InstanceContext object allows a Java component to influence the outcome of the transaction in which it is participating.
- com.sybase.jaguar.beans.enterprise.ServerBean interface – Interface for EAServer Java components, with methods that support transactional behavior and reuse of component instances.

- com.sybase.jaguar.beans.enterprise.SharedObjectException class – Class representing exceptions that are thrown by SharedObjects interface methods.
- com.sybase.jaguar.beans.enterprise.SharedObjects interface – Interface to support sharing data between instances of the same component.

# com.sybase.jaguar.beans.enterprise.EnterpriseBeanException class

| | |
|---|---|
| Description | package com.sybase.jaguar.beans.enterprise;<br>public class JCM extends Exception<br>Exception that can be thrown by components that implement the ServerBean interface. |
| Constructors | Same as java.lang.Exception. |
| Methods | Same as java.lang.Exception. |
| See also | ServerBean |

# com.sybase.jaguar.beans.enterprise.InstanceContext interface

| | |
|---|---|
| Description | package com.sybase.jaguar.beans.enterprise;<br>public interface InstanceContext extends Object<br>An InstanceContext object allows a Java component to influence the outcome of the transaction in which it is participating. A component method's calls to the InstanceContext state primitives also determine the component's state after the method completes. See "ServerBean lifecycle" on page 82 for more information. |
| Constructors | None. A component that implements the ServerBean interface receives an InstanceContext object as a parameter to the method activate(InstanceContext, String). A component that does not implement the ServerBean interface can call Jaguar.getInstanceContext() to obtain an InstanceContext object. |

Methods

- completeWork() – For transactional components, indicate that the transaction in which a component is participating should be committed. For any component, indicate that the instance should be deactivated.
- continueWork() – Indicate that the current component instance cannot be deactivated automatically when control returns from the current component method invocation.
- getSharedObjects() – Get a SharedObjects object that allows access to data shared among instances of a component.
- inTransaction() – Determine whether the current component instance is executing in the context of a transaction.
- isRollbackOnly() – Determine if the current transaction is doomed.
- rollbackWork() – For transactional components, indicate that the transaction in which a component is participating should be aborted and rolled back. For any component, indicate that the instance should be deactivated.

See also

com.sybase.jaguar.beans.enterprise.ServerBean interface, com.sybase.jaguar.beans.enterprise.SharedObjects interface

## InstanceContext.completeWork()

Description

For transactional components, indicate that the transaction in which a component is participating should be committed. For any component, indicate that the instance should be deactivated.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | InstanceContext |

```
public abstract void completeWork();
```

Usage

For a transactional component, completeWork() indicates that the component's contribution to the current transaction has been successfully completed. For any component, completeWork() indicates that the component instance should be deactivated when control returns from the current component method invocation.

If the component is transactional and the component instance is the initiator of the transaction (that is, it was instantiated directly by a base client), then EAServer attempts to commit the transaction. The transaction commits unless the commit is vetoed. Depending on the components that are participating, a veto can happen in any of the following ways:

- A participating Java component throws an exception from its ServerBean.deactivate() method.
- A participating C component has called JagDisallowCommit.
- A participating ActiveX component has called IObjectContext.disableCommit().

If the component instance is not the initiator of the transaction, the transaction may be rolled back when another participating instance calls rollbackWork() in addition to any of the cases listed above.

You can call completeWork(), continueWork(), and rollbackWork() many times in one method. Only the last call to execute takes effect. If you call none of these, the default behavior is that specified by continueWork().

See also

continueWork(), rollbackWork(), isRollbackOnly(), inTransaction()

## InstanceContext.continueWork()

Description

Indicate that the current component instance cannot be deactivated automatically when control returns from the current component method invocation.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | InstanceContext |

```
public abstract void continueWork();
```

Usage

Calling continueWork() indicates that the component instance should not be deactivated when the method returns. The component instance is not deactivated until one of the following happens:

- The transaction times out or the client's instance reference expires. In either case, the current transaction is rolled back.
- The transaction's root component calls completeWork() or rollbackWork(). If your component implements the ServerBean interface, it can veto the transaction by throwing an exception in the deactivate() method.

- The component instance calls completeWork() or rollbackWork() during a subsequent method invocation.

You can call completeWork(), continueWork(), and rollbackWork() many times in one method. Only the last call to execute takes effect. If you call none of these, the default behavior is that specified by continueWork().

See also completeWork(), rollbackWork(), isRollbackOnly(), inTransaction()

## InstanceContext.getSharedObjects()

Description Get a SharedObjects object that allows access to data shared among instances of a component.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | InstanceContext |

```
public abstract SharedObjects getSharedObjects();
```

See also com.sybase.jaguar.beans.enterprise.SharedObjects interface

## InstanceContext.inTransaction()

Description Determine whether the current component instance is executing in the context of a transaction.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | InstanceContext |

```
public abstract boolean inTransaction();
```

Return value `true` if the current component instance is executing as part of a transaction; `false` otherwise.

See also completeWork(), continueWork(), isRollbackOnly(), rollbackWork()

## InstanceContext.isRollbackOnly()

Description Determine if the current transaction is doomed.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | InstanceContext |

```
public abstract boolean isRollbackOnly();
```

Return value

`true` if the current transaction is doomed; `false` if the transaction is in a committable state or if the current component instance is not executing as part of a transaction.

Usage

Call isRollbackOnly() to determine whether the current transaction is still viable.

If a component participates in a multi-component transaction, you should call isRollbackOnly() in the following places:

- After issuing intercomponent calls
- At the start of methods that can be executed by intercomponent calls.

If the transaction is no longer viable, there is no point in continuing execution. The method should clean up and call rollbackWork() to deactivate the component instance.

See also

completeWork(), continueWork(), inTransaction(), rollbackWork()

## InstanceContext.rollbackWork()

Description

For transactional components, indicate that the transaction in which a component is participating should be aborted and rolled back. For any component, indicate that the instance should be deactivated.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | InstanceContext |

```
public abstract void rollbackWork();
```

Usage

For a transactional component, rollbackWork() indicates that the component cannot complete its contribution to the current transaction. After the method returns, the transaction is doomed: the transaction flow continues until all participating components are deactivated. At that point, the transaction is rolled back.

In any component, rollbackWork() indicates that the component instance should be deactivated when control returns from the current component method invocation.

You can call rollbackWork(), continueWork(), and completeWork() many times in one method; only the last call to execute takes effect. If you call none of these, the default behavior is that specified by continueWork().

Transactional components that make intercomponent method calls can call isRollbackOnly() to determine whether the current transaction is still viable or has been set to rollback only.

See also

completeWork(), continueWork(), inTransaction(), isRollbackOnly()

# com.sybase.jaguar.beans.enterprise.ServerBean interface

Description

```
package com.sybase.jaguar.beans.enterprise;
public interface ServerBean
```

Interface for EAServer Java components, with methods that support transactional behavior and reuse of component instances.

Constructors

None required. If a component's implementation class provides a default constructor, the EAServer runtime server calls the default constructor when creating a new component instance.

Methods

- activate(InstanceContext, String) – Indicates that this component instance has been activated.
- canReuse() – Specify whether this component instance is eligible for reuse.
- deactivate() – Indicates that this component instance has been deactivated.
- destroy() – Indicates that this component instance is being released and will not be activated again.

Usage

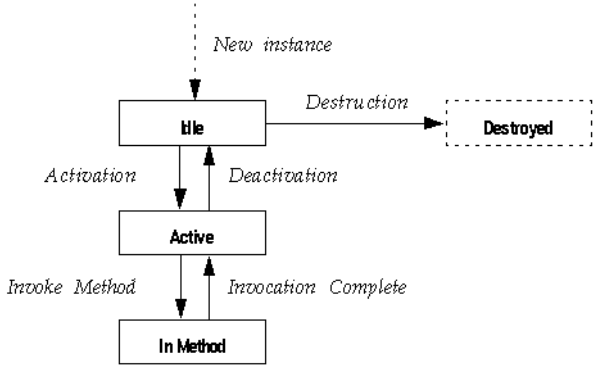
A component that implements ServerBean can participate in instance pooling. The server can maintain a cache of idle component instances and bind them to individual clients only as needed. This strategy allows the server to service more clients without the performance drain caused by allocating a component instance for each request.

The activate(InstanceContext, String) method indicates that an instance is being removed from the pool to service a client. The deactivate() method indicates that the instance is finished servicing the client. Instance reuse is optional (see “Support for instance pooling” on page 83). However, components that support it will achieve greater scalability.

### ServerBean lifecycle

Figure A-1 illustrates the states and state transitions in the lifecycle of a Java component that implements ServerBean.

***Figure A-1: States in the ServerBean lifecycle***




The state transitions are as follows:

- *New instance* – The EAServer runtime allocates a new instance of the component class. The default constructor is called if one exists. The instance remains idle until the first method invocation.
- *Activation* – Activation prepares a component instance for use by a client. activate(InstanceContext, String) is called. Once an instance is activated, it is bound to one client and can service no other client until it has been deactivated.
- *In Method* – In response to a method invocation request from the client, the EAServer runtime calls the corresponding class method in the component. The next state depends on the method’s execution, as follows:
  - If the method throws an uncaught exception, the instance is deactivated. If the method is participating in a transaction, the transaction is rolled back.

  - If the method has called InstanceContext.rollbackWork() or InstanceContext.completeWork(), the instance is deactivated.
  - If the method has called InstanceContext.continueWork(), the instance is not deactivated. The client's next method invocation is serviced by the same instance unless the client destroys its reference or disconnects.

- *Deactivation* – Deactivation occurs when:
  - The instance has called either InstanceContext.rollbackWork() or InstanceContext.completeWork()
  - The current transaction times out, or
  - The client's instance reference has expired.

  The EAServer runtime calls the component's deactivate() method to indicate deactivation.

  You can define your component so that instances are recycled after deactivation, as described in "Support for instance pooling" on page 83.

- *Destruction* – The EAServer runtime calls destroy() to indicate that references to the class instance are being released. The instance is deallocated at a later time by the Java garbage collector thread.

### Support for instance pooling

Instance pooling allows a single component instance to be activated and deactivated many times to serve different clients. Instance pooling can increase the performance of your application, since it eliminates unnecessary instance allocations. There are two ways to support pooling:

- In the Management Console, you can configure your component so instances are always pooled by selecting the Pooling option in the component properties.
- Alternatively, you can implement the ServerBean.canReuse() method to specify at runtime whether an instance can be pooled. If canReuse() returns `true`, the instance is pooled. Otherwise, the instance is destroyed.

If the component's Pooling option is enabled, EAServer never calls the canReuse() method since instances are always pooled.

If your component supports pooling, you must add code to the activate(InstanceContext, String) method that resets any class variables to their initial values. When activate returns, the component state must be the same as if the component were freshly constructed. If the component keeps references to stateful objects across activation cycles, you must reset these objects to an initial state as well.

See also InstanceContext

## ServerBean.activate(InstanceContext, String)

Description Indicate that this component instance has been activated.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | ServerBean |

```
public abstract void activate
        (InstanceContext ctx, String instanceKey)
        throws EnterpriseBeanException;
```

Parameters

*ctx*

An InstanceContext that is associated with the current component instance. activate should save a reference to the instance context for use in later method calls. This reference becomes invalid and must be discarded when deactivate() is called.

*instanceKey*

Not used.

Usage activate and deactivate allow a component's instances to be pooled. If a component supports instance pooling, activate must reset any class variables to the initial values, as if the component instance were being freshly constructed. To prohibit instance pooling, code the canReuse() method to return `false`.

See "ServerBean lifecycle" on page 82 for more information on when activate and deactivate are called.

If a component is declared to be transactional and its activate method throws an exception, the EAServer runtime server rolls back the transaction in which the component is about to participate.

See also deactivate(), canReuse()

## ServerBean.canReuse()

Description

Specify whether this component instance is eligible for reuse.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | ServerBean |

```
public abstract boolean canReuse()
```

Return value

`true` or `false` to indicate whether the component instance is eligible to be recycled.

Usage

If the Pooling option is not set in component properties, EAServer calls the component's canReuse method after deactivating each instance to determine whether the instance can be reused. If canReuse returns `false`, EAServer destroys the instance. If the Pooling option is set, EAServer never calls the canReuse method. .

Components that support instance pooling must be coded such that a recycled instance behaves the same as a newly allocated instance. Your implementation of the activate(InstanceContext, String) method must ensure that the instance state is reset to that of a newly allocated instance.

See also

activate(InstanceContext, String), deactivate(), destroy()

## ServerBean.deactivate()

Description

Indicates that this component instance has been deactivated.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | ServerBean |

```
public abstract void deactivate()
        throws EnterpriseBeanException;
```

Usage

The EAServer runtime calls deactivate() to indicate that the component instance is being deactivated. See "ServerBean lifecycle" on page 82 for more information on when activate and deactivate are called.

If your component caches data changes, you can code the deactivate() method to send cached changes to the remote database server. deactivate() can call InstanceContext.isRollbackOnly() to determine whether the current transaction is being committed or rolled back. If the transaction is being committed, deactivate() must send any cached database changes to the remote server(s).

If deactivate() throws an exception, the current transaction (if any) is rolled back; the caller of the component method that attempted to commit the transaction receives the exception as a JException with the message text included.

If your component is transactional and it maintains state (it calls InstanceContext.continueWork() from one or more methods), then deactivate() must verify that the current component state is ready for commit and throw an exception if it is not.

**Note**
deactivate should release references to the InstanceContext object that was received in the activate(InstanceContext, String) method. The InstanceContext is meaningless after deactivate has been called.

See also activate(InstanceContext, String), canReuse(), destroy()

## ServerBean.destroy()

Description Indicates that this component instance is being released and will not be activated again.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | ServerBean |

```
public abstract void destroy();
```

Usage destroy should release any resources that were allocated by the component's constructor.

See also activate(InstanceContext, String), deactivate(), canReuse()

# com.sybase.jaguar.beans.enterprise.SharedObjectException class

Description

```
package com.sybase.jaguar.beans.enterprise;

public class SharedObjectException
        extends Exception
```

Class representing exceptions that occur during SharedObjects processing.

Constructors — Same as java.lang.Exception.

Methods — Same as java.lang.Exception.

See also — SharedObjects

# com.sybase.jaguar.beans.enterprise.SharedObjects interface

Description

```
package com.sybase.jaguar.beans.enterprise;
public interface SharedObjects
```

Interface to support sharing data between instances of the same component.

Constructors — None. See InstanceContext.getSharedObjects(), ServerBean.activate(InstanceContext, String).

Methods

- get(int) – Retrieve the value of a property.
- lock(int) – Place an advisory lock on a property.
- lockNoWait(int) – Place an advisory lock on a property. If the property is currently locked, do not wait for the current lock to be released and execution immediately returns to the calling method.
- set(int, Object) – Set the value of a property.
- unlock(int) - Unlock a property locked by the same instance executing the unlock method.

See also — com.sybase.jaguar.beans.enterprise.InstanceContext interface

## SharedObjects.get(int)

Description — Retrieve the value of a property.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | SharedObjects |

```
public abstract Object get
        (int index)
        throws SharedObjectException;
```

Parameters

*index*

An arbitrary integer that identifies the property from which you want to retrieve the value.

Usage

To retrieve a property value, retrieve an object reference to the property using the get method and then assign the object reference to a variable with the desired datatype. If the property has not been initialized, the property and variable are initialized to null.

Executing a single get method on a property is atomic. *Atomic* means that an operation on data will complete before any other operations can access that data.

See also

set(int, Object), lock(int), lockNoWait(int), unlock(int)

## SharedObjects.lock(int)

Description

Place an advisory lock on a property.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | SharedObjects |

```
public abstract void lock
        (int index)
        throws SharedObjectException;
```

Parameters

*index*

An integer that identifies the property you want to lock.

Usage

Use the lock method in combination with the lockNoWait and unlock methods to synchronize multiple updates to and reads from the same property value. The lock method places an advisory lock on a property. An *advisory lock* prevents another instance from locking the property but does not prevent another instance from using the get and set methods to retrieve and update the property value. If the property is currently locked, the lock method waits for the current lock to be released.

You must lock a property before using the get or set method to retrieve or update the property value. When you lock a property that has not been set, the property is created and its value is initialized to `null`. You can lock the same property more than once as long as all locks are executed from the same component instance. However, these multiple locks are not iterative and you only have to unlock the property once.

See also lockNoWait(int), unlock(int), get(int), set(int, Object)

## SharedObjects.lockNoWait(int)

Description Place an advisory lock on a property. If the property is currently locked, do not wait for the current lock to be released and execution immediately returns to the calling method.

Syntax

| Package | com.sybase.jaguar.beans.enterprise |
|---|---|
| Interface | SharedObjects |

```
public abstract void lockNoWait
        (int index)
        throws SharedObjectException;
```

Parameters *index*

An integer that identifies the property you want to lock.

Usage Use the lockNoWait method in combination with the lock and unlock methods to synchronize multiple updates to and reads from the same property value. The lockNoWait method places an advisory lock on a property. An *advisory lock* prevents another instance from locking the property but does not prevent another instance from using the get and set methods to retrieve and update the property value. If the property is currently locked, the lockNoWait method does not wait for the current lock to be released and execution immediately returns to the calling method.

You must lock a property before using the get or set method to retrieve or update the property value. When you lock a property that has not been set, the property is created and its value is initialized to `null`. You can lock the same property more than once as long as all locks are executed from the same component instance. However, these multiple locks are not iterative and you only have to unlock the property once.

See also lock(int), unlock(int), get(int), set(int, Object)

## SharedObjects.set(int, Object)

Description

Set the value of a property.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | SharedObjects |

```
public abstract Object set
        (int index)
        Object obj)
       throws SharedObjectException;
```

Parameters

*index*

An integer that identifies the property for which you want to set a value.

*obj*

An object containing the new property value.

Usage

To set a property value, assign a value an object and pass that object as the *obj* parameter in the set method.

Executing a single set method on a property is atomic. That is, the call will complete before any other operations can access the property being set.

See also

get(int), lock(int), lockNoWait(int), unlock(int)

## SharedObjects.unlock(int)

Description

Unlock a property locked by the same instance executing the unlock method.

Syntax

| | |
|---|---|
| Package | com.sybase.jaguar.beans.enterprise |
| Interface | SharedObjects |

```
public abstract void unlock
        (int index)
        throws SharedObjectException
```

Parameters

*index*

An integer that identifies the property to be locked.

Usage

Use the unlock method in combination with the lock and lockNoWait methods to synchronize multiple updates to and reads from the same property value. The unlock method releases an advisory lock on a property that has been locked by the instance executing the unlock method. An *advisory lock* prevents another instance from locking the property but does not prevent another instance from using the get and set methods to retrieve and update the property value.

You can unlock a property that has not been set. Even if a property has been locked more than once, you only have to unlock the property once.

See also

lock(int), lockNoWait(int), get(int), set(int, Object)

# Index

## B



## C

## D



## F



## G



## I



## J

## N



## R



## S

## T



## W