jCaoSQL

User's guide

version 1.01

April 24, 2015

[Remarks]

[Revision history]

Date	Ver.	Description
2006/02/27	1.0	First edition
2015/04/24	1.01	Added RMI relation function

Contents

1. Introduction 5

2. System structure 6

2.1. System requirements 6
2.2. Class structure 6

3. Development environment setup 7

3.1. Development environment 7
3.2. Development environment setup 7
3.2.1. Java environment setup 7
3.2.2. VC++ environment setup 7
3.3. Building with Ant 10
3.3.1. Ant setup 10
3.3.2. How to use Ant 10

4. jCaoSQL Feature 12

4.1. Generating and deleting instance 12
4.2. Mapping of VARIANT type 12
4.3. Event sink 13

5. Tutorial 14

5.1. Client development environment setup 14
5.2. Sample program description 14
5.2.1. Generation and discard of instance 14
5.2.2. VARIANT type 14
5.2.3. Creating an event sink 15
5.2.4. Creating a custom class 15
5.2.5. Adding and deleting items dynamically 16

6. RMI relation function 17

6.1. Overview 17
6.2. Execution environment setup 18
6.2.1. Establishing environment for ORiN server side 18
6.2.2. Establishing environment on the client side 18

6.3. Implementation 19
6.3.1. Implementation method 19
6.3.2. Sample code 21
Appendix A. Class diagram 22
Appendix B. Copyright information of Open Source Software 23

1. Introduction

This document is a user’s guide of jCaoSQL that serves as a framework to control CaoSQL from Java. With this framework, Java application can access resources, such as robot controllers, through CaoSQL interface. At present, all functions of CaoSQL can be used from Java (some functions are limited).

This document is an instruction guide to create CaoSQL client application for Java by using jCaoSQL. For details about CaoSQL interfaces, please refer to “CaoSQL external specification guide”.

2. System structure

This chapter describes jCaoSQL system structure..

2.1. System requirements

The operation of jCaoSQL is assured in the environment of Table 2-1. Although this system consists of Java language, because it uses native code in JNI, the portability between different OS is not guaranteed.

Table 2-1 Operation environment of jCaoSQL

OS	Windows7
Java	JDK 1.7.x
CaoSQL	version 1.0

2.2. Class structure

jCaoSQL corresponds with COM object model of CaoSQL. jCaoSQL has classes described in Table 2-1.

Since the function of CaoSQLHistory is to refer to the database only, there is no need of using CaoSQL interface. Therefore, jCaoSQL does not support it. If there are any needs of referring database, use any general purpose O/R mapping framework.[1]

Table 2-2 Object model correspondence between CaoSQL and CaoSQL

CaoSQL object model	jCaoSQL object model
CaoSQLEngine	jp.co.denso.fa.jcaosql.CaoSQLEngine
CaoSQLController	jp.co.denso.fa.jcaosql.CaoSQLController
CaoSQLItem	jp.co.denso.fa.jcaosql.CaoSQLItem
CaoSQLHistory	-

For details about class diagram of jCaoSQL, refer to Appendix A.Class diagram.

[1] Major mapping frameworks are ; Apache Torque, Apache Commons DbUtil, (Castor) JDO, and Hibernate.

3. Development environment setup

This chapter describes jCaoSQL development setup. These settings are not required unless you refer or modify jCaoSQL code directory.

3.1. Development environment

jCaoSQL has been developed in the environment shown in Table 3-1. This environment is required to build from source code.

Table 3-1 Development environment of jCaoSQL

OS	Windows7
Java	JDK 1.7.x
C++	Microsoft Visual C++2005
Build tool	Apache Ant 1.6.2
Documentation generator	Doxygen 1.2.18 + Graphviz 1.10
Testing framework	JUnit 3.8.1
CaoSQL	version 1.0

3.2. Development environment setup

Perform the following steps for respective environment settings.

3.2.1. Java environment setup

From the official website of Oracle, download JDK 1.7.x, and then install it in a computer that you use for development.

On environment variable of the computer, specify a JDK installation folder as **JAVA_HOME.**

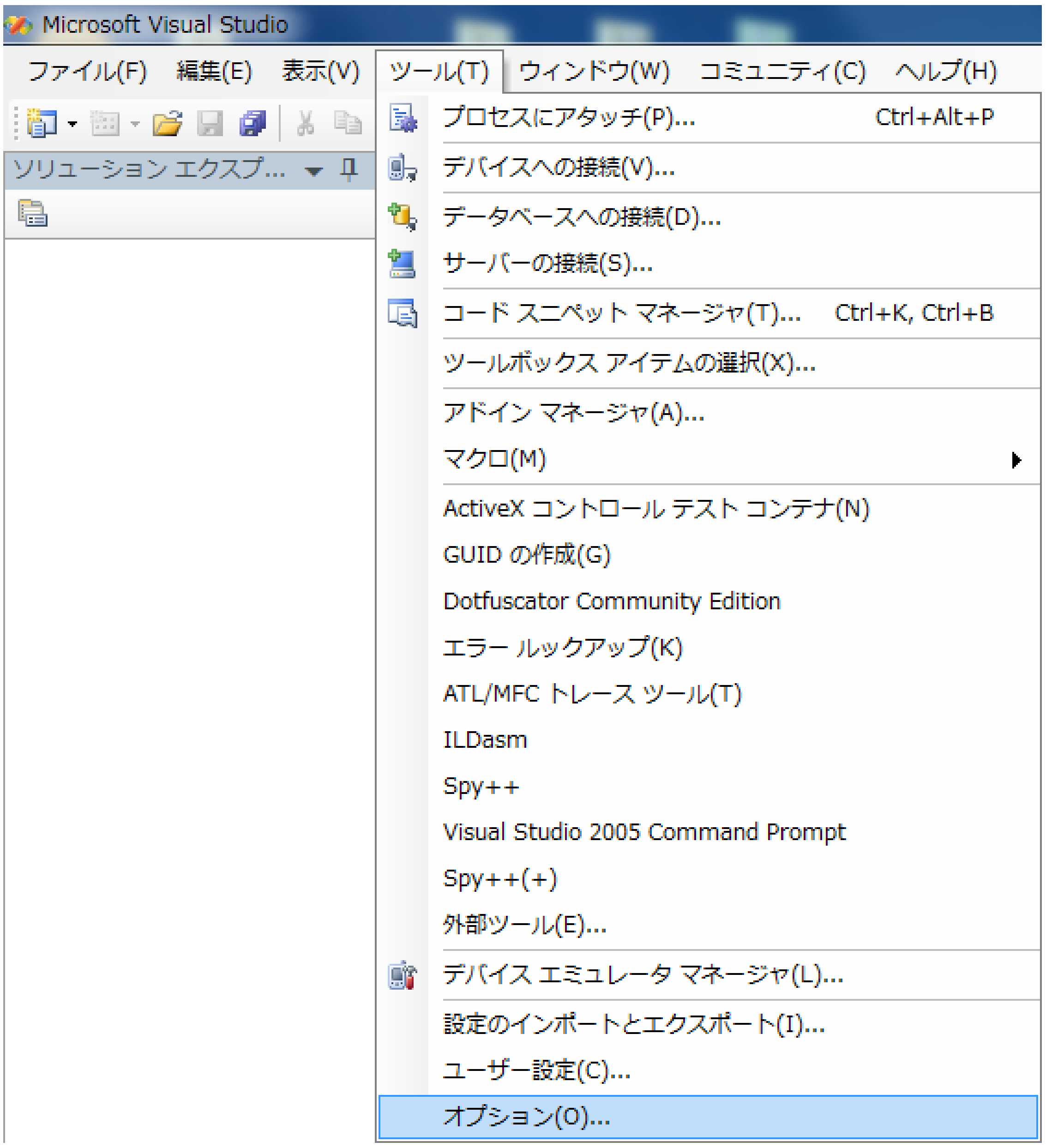
Add %**JAVA_HOME**%¥bin to PATH of environment variable.

From the official website of JUnit, download JUnit3.8.1. Unpack the downloaded file. On environment variable of the computer, specify "junit.jar", which is stored in the unpacked folder, as **CLASSPATH**.

3.2.2. VC++ environment setup

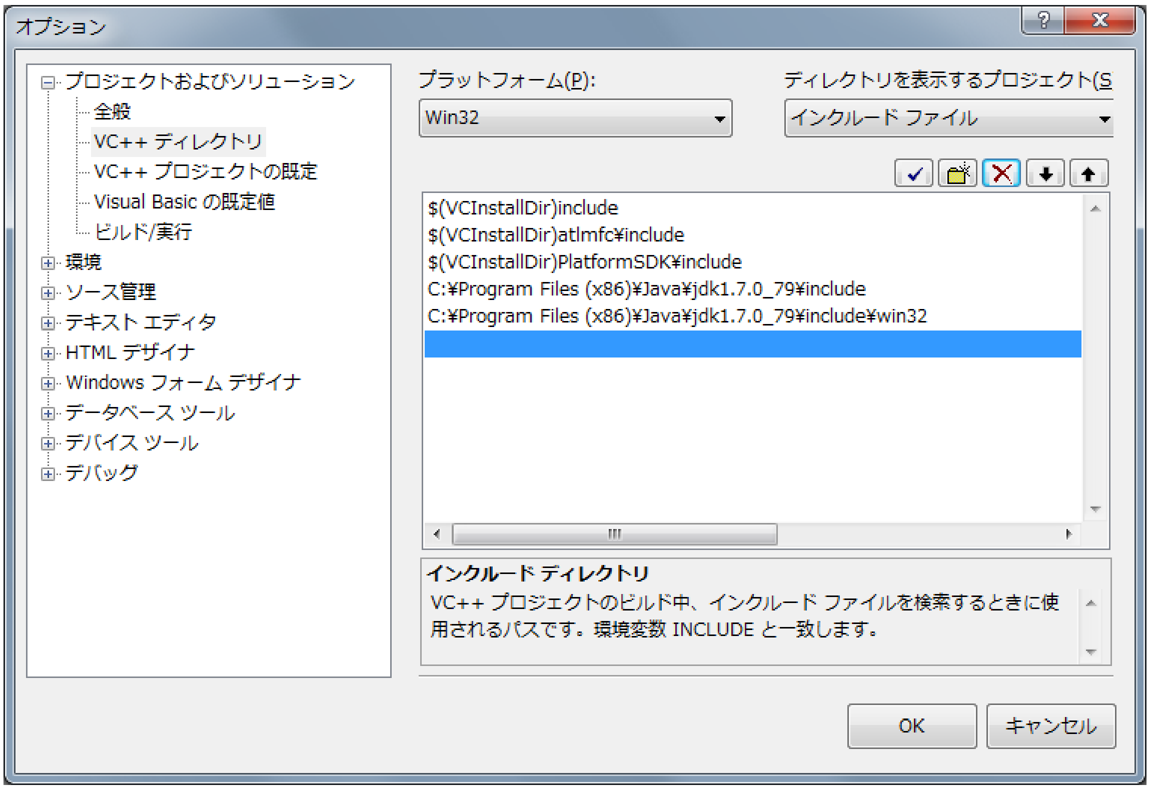
For the preparation of JNI compile environment, do the following settings.

From the menu bar of VC, click Tool, click Option.


Microsoft Visual Studio
ファイル(F)
編集(E)
表示(V)
ツール(T)
ウィンドウ(W)
コミュニティ(C)
ヘルプ(H)
ソリューション エクスプ...
プロセスにアタッチ(P)... Ctrl+Alt+P
デバイスへの接続(V)...
データベースへの接続(D)...
サーバーの接続(S)...
コード スニペット マネージャ(T)... Ctrl+K, Ctrl+B
ツールボックス アイテムの選択(X)...
アドイン マネージャ(A)...
マクロ(M)
ActiveX コントロール テスト コンテナ(N)
GUID の作成(G)
Dotfuscator Community Edition
エラー ルックアップ(K)
ATL/MFC トレース ツール(T)
ILDasm
Spy++
Visual Studio 2005 Command Prompt
Spy++(+)
外部ツール(E)...
デバイス エミュレータ マネージャ(L)...
設定のインポートとエクスポート(I)...
ユーザー設定(C)...
オプション(O)...

From the **Platform** drop-down list, select “Win32”, from the **Show directories for** drop-down list, select “Include files”, and then add %**JAVA_HOME**%¥include and %**JAVA_HOME**%¥include¥win32 to the directory list. [2]

[2] Replace the environment variable of %JAVA_HOME% to the variable that you have specified in 3.2.1.



オプション

- プロジェクトおよびソリューション
  - 全般
  - VC++ ディレクトリ
  - VC++ プロジェクトの既定
  - Visual Basic の既定値
  - ビルド/実行
- 環境
- ソース管理
- テキスト エディタ
- HTML デザイナ
- Windows フォーム デザイナ
- データベース ツール
- デバイス ツール
- デバッグ

プラットフォーム(P): Win32

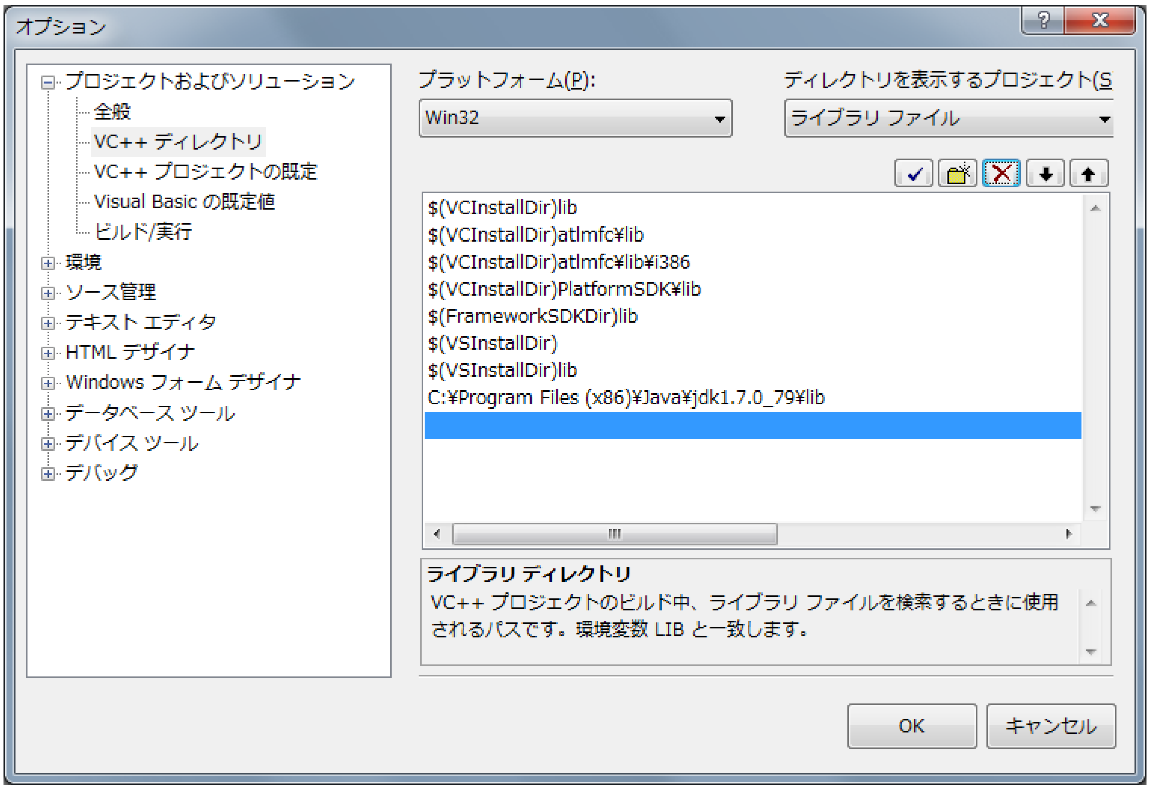
ディレクトリを表示するプロジェクト(S: インクルード ファイル

```
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include
C:¥Program Files (x86)¥Java¥jdk1.7.0_79¥include
C:¥Program Files (x86)¥Java¥jdk1.7.0_79¥include¥win32
```

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

OK キャンセル

From the **Show directories for** drop-down list, select “Library files”, and then add %**JAVA_HOME**%¥lib to the directory list[3].



オプション

- プロジェクトおよびソリューション
  - 全般
  - VC++ ディレクトリ
  - VC++ プロジェクトの既定
  - Visual Basic の既定値
  - ビルド/実行
- 環境
- ソース管理
- テキスト エディタ
- HTML デザイナ
- Windows フォーム デザイナ
- データベース ツール
- デバイス ツール
- デバッグ

プラットフォーム(P): Win32

ディレクトリを表示するプロジェクト(S: ライブラリ ファイル

```
$(VCInstallDir)lib
$(VCInstallDir)atlmfc¥lib
$(VCInstallDir)atlmfc¥lib¥i386
$(VCInstallDir)PlatformSDK¥lib
$(FrameworkSDKDir)lib
$(VSInstallDir)
$(VSInstallDir)lib
C:¥Program Files (x86)¥Java¥jdk1.7.0_79¥lib
```

ライブラリ ディレクトリ
VC++ プロジェクトのビルド中、ライブラリ ファイルを検索するときに使用されるパスです。環境変数 LIB と一致します。

OK キャンセル

[3] Replace the environment variable of %JAVA_HOME% to the variable that you have specified in 3.2.1

3.3. Building with Ant

3.3.1. Ant setup

From the website of Apache Ant, download Ant 1.6.2 and expand archives.

On environment variable of the computer, specify expanded ANT folder as **ANT_HOME**.

Add %ANT_HOME%¥bin to **PATH** of environment variable.

3.3.2. How to use Ant

3.3.2.1. Individual settings

jCaoSQL equips build.properties file to configure individual property of development environment. Modify the following two properties according to the environment you use.

```
caosql.src=[Source directory of CAOSQL]
msvc.path=[Path to Visual C++]
```

Note that the path delimiter used here should be backslash ('/') instead of '¥'.

3.3.2.2. About Ant tasks

jCaoSQL project provides the following Ant tasks. Use these tasks according to your needs.

Table 3-2 Ant tasks available in jCaoSQL

Task name	Description
clean	Delete a work directory
init	Create a work directory
jni	Create a JNI header
compile	Execute compile-java and compile-native tasks
compile-java	Compile java source
compile-native	Compile native code of C++
dll	Execute jni and compile-native tasks
rebuild	Execute clean and compile tasks
Test	Execute all unit tests and create reports
suitetest	Execute specified testsuite and create reports (Specify a testsuite with "-Dtest.class=[test suite]")
xdoc	Execute javadoc and doxygen tasks
javadoc	Create API documents in Java code
doxygen	Create API document in native code
jar	Create jar archive
dist	Create dll and jar documents and allocate them to dist folder

3.3.2.3. Executing Ant tasks

To execute a task, on the command line, enter “*ant [task name]*”.

If a task requires arguments, enter arguments after the task name.

ant [task name] (argument)

For example, if you execute only “CaoSQLTest” in suitetest, write as follows.

ant suitetest –Dtest.class=CaoSQLTest

4. jCaoSQL Feature

4.1. Generating and deleting instance

Table 2-2 shows the object model correspondence between jCaoSQL and CaoSQL. Java objects in this table generate COM interfaces, control interface pointers, and manage reference counters.[4]

Only CaoSQLEngine has a factory method[5] that can generate an instance. Other object should be generated as a child object of Engine so that it can obtain an interface pointer. In other word, these objects should be generated by AddController. If an object is generated any other way, the created object is not included in the object tree (which means the object cannot obtain an interface pointer), as a result, the object cannot be controlled by CaoSQL.

If an object A is referred by another object B which has an interface pointer, the object A uses AddRef and Release methods to control the reference counter that the object B has. When referring objects, make sure that the counter will be "0" at the end of reference. If the counter remains "1" or higher after the reference end, CaoSQL's COM object remains, resulting in memory leaks.

For details about instance management, refer to the sample program in 5.2.1.

4.2. Mapping of VARIANT type

COM has a VARIANT type which is a general-purpose data type. VARIANT type is used for entering tags and/or Item's values in CaoSQL as well. In Java, because there is no data type that corresponds to VARIANT type, jCaoSQL maps such data to java.variant.Object type. Note that not all data are allocated to VARIANT type. The following table shows available data type.

Table 4-1 VARIANT mapping

VAIANT type	Description
VT_BOOL	java.lang.Boolean
VT_I1	java.lang.Character
VT_UI1	java.lang.Byte
VT_I2(VT_UI2)	java.lang.Short
VT_I4(VT_UI4)	java.lang.Integer
VT_R4	java.lang.Float
VT_R8	java.lang.Double
VT_BSTR	java.lang.String
VT_DATE	java.util.Date
VT_ARRAY | VT_BOOL	java.lang.Boolean[]

[4] Other than COM's reference counter, Java object itself has a reference counter as well. It is impossible to release more than the time that Java object has been referred.
[5] CaoSQLEngine#createInstance method

VT_ARRAY | VT_I1	java.lang.Character[]
VT_ARRAY | VT_UI1	java.lang.Byte[]
VT_ARRAY | VT_I2(VT_UI2)	java.lang.Short[]
VT_ARRAY | VT_I4(VT_UI4)	java.lang.Integer[]
VT_ARRAY | VT_R4	java.lang.Float[]
VT_ARRAY | VT_R8	java.lang.Double[]
VT_ARRAY | VT_BSTR	java.lang.String[]
VT_ARRAY | VT_DATE	java.util.Date[]
VT_ARRAY | VT_VARIANT	java..lang.Object[]

There are some restrictions.

・ Since Java does not support unsigned short(VT_UI2) and long(VT_UI4), jCaoSQL stores such values as signed when receiving. Also, jCaoSQL cannot set values of unsigned short and long (receive only).

・ Array type supports one dimensional array only. Using two or more dimensional array results in an exception type error.[6]

・ VT_VARIANT supports an array which elements includes two or more data types; for example, Object[] array = {new Integer(1), new String("aa")}. This Object cannot nest another Object.

・ Because Java does not have Currency(VT_CY) type, use java.text.NumberFormat to express currency type.

4.3. Event sink

jCaoSQL can callback CaoSQL's Lisner object when an event generates in CaoSQL. To do so, create a class that implements CaoSQLControllerListener interface, and pass it to attachListener of CaoSQLController. To terminate event reception, call dettachListener. This processing receives an event by attachListener, just like the way of instance management. CaoSQLController needs to call dettachListener when object is discarded. For details about implementation, refer to the sample program in 5.2.3.

[6] jp.co.denso.fa.UnSupportedTypeException will occur.

5. Tutorial

5.1. Client development environment setup

This chapter describes how to develop client by using jCaoSQL. To develop a client, following files in jCaoSQL project are used.

Table 5-1 Files in jCaoSQL

File name	Description
	How to set
jCaoSQL.jar	Java binary package of jCaoSQL
	Through ClassPath
jCaoSQL.dll	Native code of jCaoSQL
	Allocate this file to the directory where client can refer, or register the directory to %PATH% of environment variable.
log4cpp.dll	Library to output log information
	Register the directory to %PATH% of environment variable
jcaosql_log.config	Log setting information of native code
	This is option. If log information of native code is required, allocate this file to the application execution folder.

For about concrete development method, refer to the following descriptions and sample folders that are added to the package.

5.2. Sample program description

5.2.1. Generation and discard of instance

For CaoSQLEngine, CaoSQLController, and CaoSQLItem objects, respective factory methods are prepared ; CaoSQLEngine#createInstance, CaoSQLEngine#getController, and CaoSQLController#getItem. Once a factory method is called, a corresponding COM instance is generated. When you discard such objects, be sure to call “release” at any time. Since Java does not have the concept of “destructor”, you need to call “release” explicitly on a client application.

This sample is stored in “sample¥sample1” folder. You can execute the sample by using the following command from the sample folder.

ant sample1

5.2.2. VARIANT type

jCaoSQL maps VARIANT class to java.lang.Object class. For details about correspondence, refer to Table

4-1. This subsection explains how to distinguish VT type data on Java-client side.

In Java, by using instanceof operator, users can check if the target value belongs to the specified object type. In the following example, VT_I4 type VARIANT is mapped to java.lang.Integer class. Therefore, writing the following program will show VT_I4 as execution result.

```
if (value instanceof Integer) {
        // VT_I4 type variable
}
```

Also, to check whether an object is Array type, obtain java.lang.Class from the object, and call isArray method of the Class object.

This sample is stored in “sample¥sample2” folder. You can execute the sample by using the following command from the sample folder.

ant sample2

5.2.3. Creating an event sink

As the sample program shows, jCaoSQL can receive an event. However, because CaoSQL generates events asynchronously, in order to synchronize events at Java object side, you need to declare OnChangeItem method or OnChangeState method at the implementation of jp.co.denso.fa.CaoSQLControllerListener, or declare synchronized in a method in any time.

Also, a CaoSQLItem object passed as argument at OnChangeItem event must be discarded (explicitly call “release”), just like 5.2.1.

This sample is stored in “sample¥sample3” folder. You can execute the sample by using the following command from the sample folder.

ant sample3

5.2.4. Creating a custom class

To make design and implementation more convenient, you can create a custom class of CaoSQLjava object that implements original functions. To do so, define a custom class in jcaosql.properties and create a derived class of existing CaoSQL.java object, as the following shows.

Table 5-2 Key list of jcaosql.properties

Key name	Description
caosql.engine.iid	Interface ID of CaoSQLEngine (no need of change in normal operation)
caosql.engine.progid	Program ID of CaoSQLEngine (no need of change in normal operation)

caosql.engine.class	CaoSQLEngin class that a factory will create. Default: jp.co.denso.fa.jcaosql.CaoSQLEngine
caosql.controller.class	CaoSQLController class that a factory will create. Default: jp.co.denso.fa.jcaosql.CaoSQLController
caosql.item.class	CaoSQLItem class that a factory will create. Default: jp.co.denso.fa.jcaosql.CaoSQLItem

For example, to use com.example.MyCaoSQLItem class which base class is CaoSQLItem, create a file “jcaosql.properties”. For <caosql.item.class> key, enter the value of <com.example.MyCaoSQL>, and then allocate the properties file on the execution route. With this operation, a factory class can generate com.example.MyCaoSQLItem class at the creation of CaoSQLItem.

This sample is stored in “sample¥sample4” folder. You can execute the sample by using the following command from the sample folder.

ant sample4

5.2.5. Adding and deleting items dynamically

Just like CaoSQL, jCaoSQL can add items dynamically, and only dynamically added items can be deleted.

This sample is stored in “sample¥sample5” folder. You can execute the sample by using the following command from the sample folder.

ant sample5

6. RMI relation function

6.1. Overview

To access CaoSQL.exe from non-Windows OS, such as Linux, use RMI relation function.

The following figure compares the mechanism to access CaoSQL.exe from Windows OS and from Linux OS.

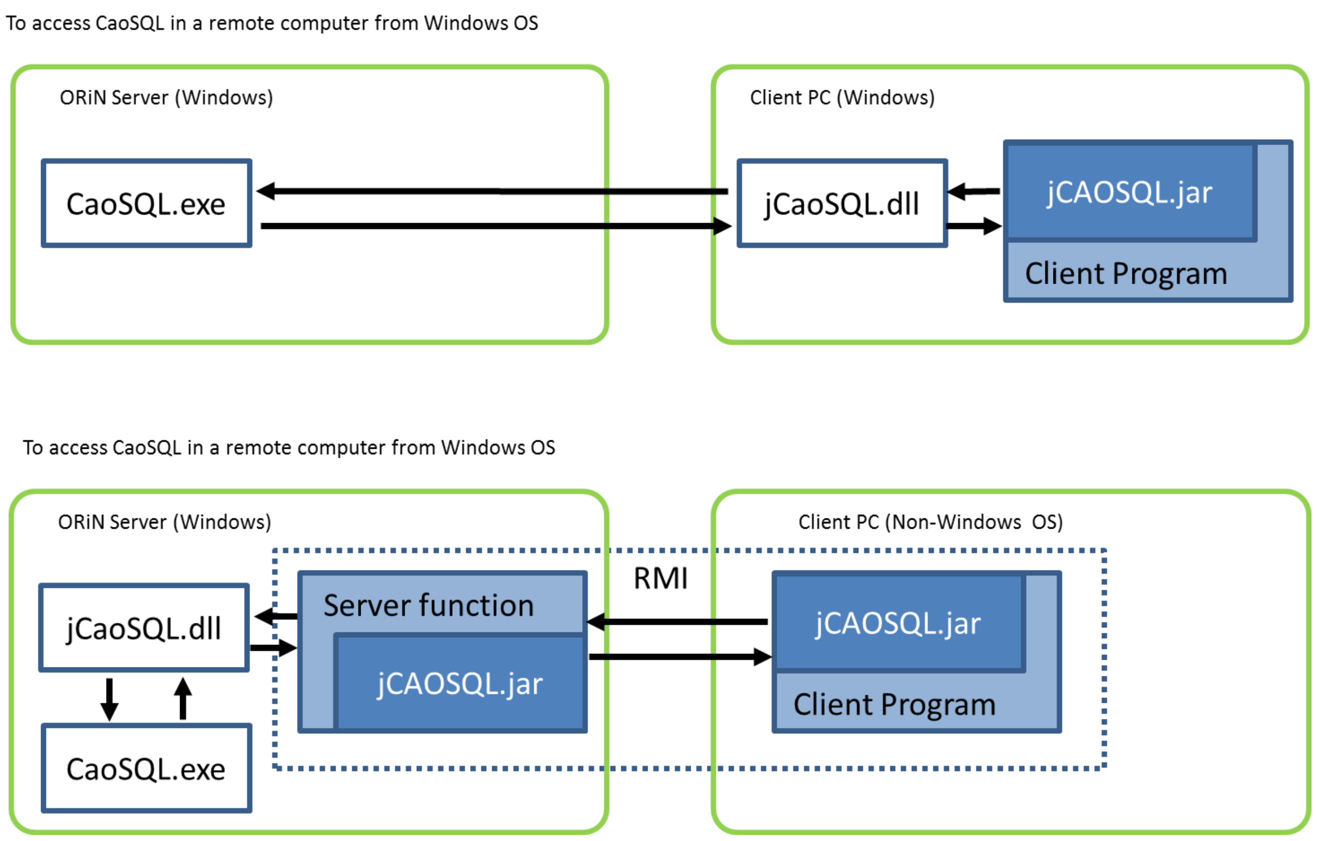


Figure 6-1 Concept of RMI relation function

jCaoSQL consists of jCaoSQL.jar and jCaoSQL.dll in order to communicate with CaoSQL.exe through COM communication.

Because non-Windows OS cannot execute dll, to access CaoSQL, use jCaoSQL.jar as a server in order to run a function of jCaoSQL.jar in remote.

Users can switch programs of the client side to non-Windows OS-dedicated Java application by changing package used.

Some implementation needs to be modified. For details, refer to 6.3.1.

6.2. Execution environment setup

6.2.1. Establishing environment for ORiN server side

Execute jCaoSQLServer.cmd by using files in Table 6-1.

Table 6-1 Files in jCaoSQL necessary for environment setup (ORiN server side)

File name	Description / How to set
	How to set
jCaoSQL.jar	Java binary package of jCaoSQL
	Allocate this file to the directory where jCaoSQLServer.cmd can refer, or register the directory to %**PATH**% of environment variable.
jCaoSQL.dll	Native code of jCaoSQL
	Allocate this file to the directory where jCaoSQL.jar can refer, or register the directory to %**PATH**% of environment variable.
policy.txt	Security policy to start server function
	Allocate the directory same as jCaoSQLServerStart.cmd
jCaoSQLServer.cmd	Batch file to start server function
	Allocate this file to the application execution folder. Specify the directory path to "jCaoSQL.jar" to -Djava.rmi.server.codebase, and specify the directory path to "policy.txt" to -Djava.security.policy.
log4cpp.dll	Library to output log information
	Register the directory to %**PATH**% of environment variable
jCaosql_log.config	Log setting information of native code
	This is option. If log information of native code is required, allocate this file to the application execution folder.

6.2.2. Establishing environment on the client side

Implement client program by using files in Table 6-2.

Table 6-2 Files in jCaoSQL necessary for environment setup (client side)

File name	Description
	How to set
jCaoSQL.jar	Java binary package of jCaoSQL
	Pass through ClassPath.

6.3. Implementation

6.3.1. Implementation method

RMI relation function invokes functions on ORiN server side in remote by using Java RMI. Please note that the following points when implementation

- **Library reference**

  The following two files must be the same version in both ORiN server and Client sides.

  ・jCaoSQL.jar

  ・policy.txt

  ※It is recommended to store these files in the directory which is accessible from both ORiN server side and client side.

- **Package used**

  To access CaoSQL.exe by using RMI relation function, use API written in jp.co.denso.fa.jcaosql.rmi or below. Also, you can use an existing program by changing the package used (by rewriting import sentence).

  (Example)

  ```
  // import jp.co.denso.fa.jcaosql.*;
  import jp.co.denso.fa.jcaosql.rmi.*;
  ```

  ※Please note that the same class name are used.

- **CaoSQLEngine.createInstance() method**

  For argument, specify “Conn = <RMI server name (ORiN server’s IP address or computer name)”.

  (Example) To connect localhost

  ```
  CaoSQLEngine.createInstance(“Conn=localhost”)
  ```

- **JavaVM security policy settings**

  Pass "policy.txt" as an argument at VM execution, or set "policy.txt" to Java security policy file (%JAVA_HOME%/jre/lib/security/java.policy) to be executed.

  ✧ To pass "policy.txt" as an argument at Java VM execution

  Specify the following line to the argument.

  ```
  -Djava.security.manager -Djava.security.policy=《path of policy.txt》
  ```

  ✧ To set Java security policy file

  Add the following lines to %JAVA_HOME%/jre/lib/security/java.policy.

  ```
  grant {
          permission java.lang.RuntimePermission "loadLibrary.jCaoSQL";
          permission java.lang.RuntimePermission "createSecurityManager";
          permission java.lang.RuntimePermission "setSecurityManager";
          permission java.util.PropertyPermission "user.dir", "read";
          permission java.util.PropertyPermission
                  "java.rmi.server.hostname", "read";
          permission java.net.SocketPermission
                  "< RMI server name (ORiN server's IP address or computer
                  name)>:1024-65535","connect,accept";
          permission java.net.SocketPermission
                  "<Client name>:1024-65535","connect,accept";
          };
  ```

  ※Security policy is required on both the server and client sides.

- **Execution of server function**

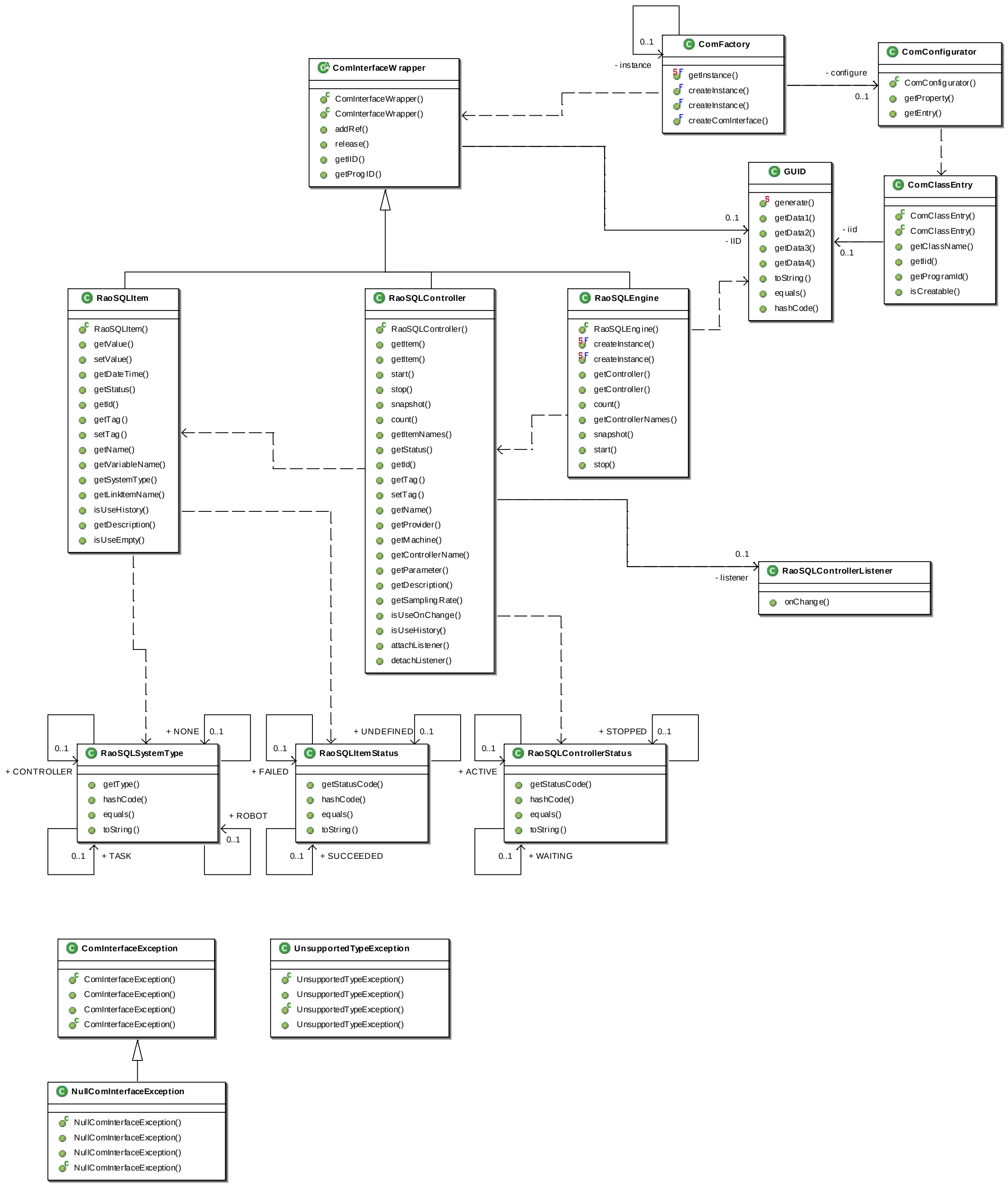
  Execute jCaoSQLServer.cmd on ORiN server side.

※Other points are the same as general jCaoSQL specifications. For details, refer to 4.jCaoSQL Feature.

6.3.2. Sample code

```
public class Main {
	public static void main(String[] args) {
		CaoSQLEngine eng = null;
		CaoSQLController ctrl = null;
		CaoSQLItem item = null;
		try {
			eng = CaoSQLEngine.createInstance("localhost");    //Obtain CaoSQLEngine instance
			String[] ctrl_names = eng.getControllerNames();    //Obtain Controller name list
			ctrl = eng.Controller(ctrl_names[0]);    //Obtain CaoSQLController instance
			String[] item_names = ctrl.getItemNames();    //Obtain Item name list
			item = ctrl.Item(item_names[0]);    //Obtain CaoSQLItem instance
			Object value = new Integer(100);
			item.putValue(value);    //Write a value into Item
			//Obtain setting values from Item
			System.out.println("VariableName: " + item.getSettingData(new Integer(0)));
			//Obtain values from Item
			System.out.println("Value: " + item.getValue().toString());
			//Obtain value update timing from Item
			System.out.println("Updated: " + item.getDateTime().toString());
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			if (item != null) item.release();
			if (ctrl != null) ctrl.release();
			if (eng != null) eng.release();
		}
	}
```

Appendix A. Class diagram



Appendix B. Copyright information of Open Source Software

jCaoSQL.dll uses open source software “log4cpp”. The following shows copyright information.