The Omnis Java API

The Omnis Java API

Introduction

This document describes the Omnis Java API that provides an interface to Java for Omnis External Component developers. A familiarity with the Omnis External Component interface is required before reading this document and a good knowledge of the EXTfldval and EXTqlist classes is essential. Knowledge of the Java Language would also be helpful.

Requirements

In order to use Omnis with Java, you will need either the Sun Java 2 Runtime Environment (J2SE) or the Sun Java 2 SDK installed and configured correctly on your system. OSX users do not have to worry as this OS comes with Java 2 pre-installed. Linux and Solaris users should make sure that their $LD_LIBRARY_PATH environment variable is configured to point at the Java Virtual Machine hotspot/client engine (libjvm.so).

The Omnis Java Engine (OJE)

The Omnis Java Engine (OJE) is responsible for all communication between Java and Omnis and is capable of maintaining a Java Session for each Omnis task instance. Each Java Session is capable of holding a variety of Java classes and objects that can be created via the APIs listed in the section entitled “Java External Component APIs”. In addition to the “default” sessions that are available to all external components running in the context of the current task instance, additional Sessions can be created (if required) that are private to your external component.

The Omnis Java Engine consists of two parts. The first part is implemented as an Omnis external component and is responsible for converting data between Omnis and Java. The second part is implemented in Java and is responsible for managing Java Sessions and maintaining a list of Java objects and classes for each Session.

Communicating Between Omnis and Java

Communication between Omnis and Java is achieved via the use of EXTfldvals and EXTqlists. These classes are used in a constructive manner to organise data and data types into an Omnis list which is then converted into a list of Java parameters by the Omnis Java Engine. These Java parameters can then be used to either construct a java object or call a java method.

The Omnis Java List Structure

When passing parameters to Java, it is important to construct a list that is formatted correctly so that it is understood and translated by the Omnis Java Engine. The format of this list structure is as follows:

Data Column	Type Column
EXTfldval containing the parameter data	EXTfldval containing the parameter type

The Data Column

The Data Column should contain a list of EXTfldvals. Each EXTfldval should be a parameter that you wish to pass to Java.

The Type Column

The Type Column should contain a list of EXTfldvals. Each EXTfldval should describe the type of the Java function parameter that you are going to call. In order to achieve this, each EXTfldval should contain a string which represents a Java Type that is supported by the Omnis Java Engine. A list of supported Java types and their associated Omnis Java Engine Type Strings (OJETS) can be seen below:

Java Basic Types	Omnis Java Engine Type String (OJETS)
int	Integer
long	Long
short	Short
float	Float
double	Double
Boolean	Boolean
char	Char
byte	Byte
int[]	[Integer
long[]	[Long
short[]	[Short
float[]	[Float
double[]	[Double
boolean[]	[Boolean
char[]	[Char
byte[]	[Byte
java.lang.Integer	Cinteger
java.lang.Long	Clong
java.lang.Short	Cshort
java.lang.Float	Cfloat
java.lang.Double	Cdouble
java.lang.Boolean	Cboolean
java.lang.Character	Cchar
java.lang.Byte	Cbyte
java.lang.String	Cstring
java.util.Date	Cdate
java.lang.Object	Cobject
Java.util.Vector	Cvector
java.lang.Integer[]	[cInteger
java.lang.Long[]	[cLong
java.lang.Short[]	[cShort
java.lang.Float[]	[cFloat
java.lang.Double[]	[cDouble
java.lang.Boolean[]	[cBoolean

java.lang.Character[]	[cChar
java.lang.Byte[]	[cByte
java.lang.String[]	[cString
java.util.Date[]	[cDate
java.lang.Object[]	[cObject

Note: OJETS are not case sensitive.

Mapping Omnis Types to Java Types

When passing parameters to Java, it is important to know which Omnis Data Types are compatible with which Java Data Types. This is illustrated in the following table:

Omnis Data Type	Java Data Type
fftInteger	int long short java.lang.Integer java.lang.Long java.lang.Short
fftNumber	float double java.lang.Float java.lang.Double
fftCharacter	char java.lang.Character char[] java.lang.Character[] java.lang.String
fftBinary	byte java.lang.Byte byte[] java.lang.Byte[]
fftDate	java.util.Date

The above shows each Omnis Data Type together with the Java Data Types that the Omnis Type can be converted to. The “Omnis Java Engine Type String” that is specified in the “Type Column” of the Omnis Java List Structure directly controls this conversion.

Example:

If you have an Integer parameter in Omnis and you wish to call a Java Method which requires java.lang.Long, your list would appear as follows:

Data Column	Type Column
EXTfldval (fftInteger) containing 150	EXTfldval containing “cLong”

The same value could be passed to a Java Method expecting a short by using the following list:

Data Column	Type Column
EXTfldval (fftInteger) containing 150	EXTfldval containing “Short”

NOTE: The Omnis Java Engine only supports the conversions shown in the “Mapping Omnis Types to Java Types” table. For example, the following will produce an error:

Data Column	Type Column
EXTfldval (fftInteger) containing 150	EXTfldval containing “Float”

The above will fail because the Data Column contains an EXTfldval of type fftInteger, which cannot be converted to a Java Float directly.

Parameters

Creating a Simple Parameter List

The example code shown below creates a parameter list, which is suitable for calling the following Java Method:

```
Public void myfunc(Integer p1, Integer p2, Float p3);
{
…
}
```

The following C++ will create an EXTqlist that is suitable for calling myfunc.

```
EXTqlist *llist = NULL;
EXTfldval p1,p2,p3,paramType1,paramType2, paramType3;
str15 pstype(QTEXT("cInteger"));
str15 pstype1(QTEXT("cFloat"));

paramType1.setChar(pstype);
paramType2.setChar(pstype);
paramType3.setChar(pstype1);

p1.setLong(50); // Parameter 1, will be converted to java.lang.Integer
p2.setLong(100); // Parameter 2, will be converted to java.lang.Integer
p3.setNum(23.215); // Parameter 3, will be converted to java.lang.Float

llist = new EXTqlist(listVlen);
llist->addCol(fftNone,dpDefault);
llist->addCol(fftNone,dpDefault);

llist->insertRow();
llist->putColVal(1, 1, p1);
llist->putColVal(1, 2, paramType1);

llist->insertRow();
llist->putColVal(2, 1, p2);
llist->putColVal(2, 2, paramType2);

llist->insertRow();
llist->putColVal(3, 1, p3);
llist->putColVal(3, 2, paramType3);

//The following code uses the above list to call myfunc. For a further explanation of
//these APIs, see the section entitled "Java External Component APIs".

jsessionid tasksession;
jobjid myobj;
str255 clsname(QTEXT("test.store.Hello"));
str255 clspath(QTEXT("C:\\myJava\\"));
str255 mymethod(QTEXT("myfunc"));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
        errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,llist);
```

Creating a Complex Parameter List

A complex parameter list can be used to call Java Methods that use arrays as parameters. Consider the following Java function:

```
Public void myfunc(Integer p1[], Float p2[])
{
…
}
```

In order to call the above function, nested EXTqlists must be used as follows:

```
EXTqlist *paramlist = NULL;
EXTqlist *intarray = NULL;
EXTqlist *floatarray = NULL;
EXTfldval p10,p11,p20,p21,paramType10,paramType11, paramType20, paramType21;
EXTfldval paramTypeIntArr, paramTypeFloatArr, element1, element2;
str15 pstype(QTEXT("cInteger"));
str15 pstype1(QTEXT("cFloat"));
str15 pstype1Arr(QTEXT("[cInteger"));
str15 pstype2Arr(QTEXT("[cFloat"));

// Create the Integer Array.
paramType10.setChar(pstype);
paramType11.setChar(pstype);
paramType20.setChar(pstype1);
paramType21.setChar(pstype1);
paramTypeIntArr.setChar(pstype1Arr);
paramTypeFloatArr.setChar(pstype2Arr);


p10.setLong(50); // Parameter 1[0], will be converted to java.lang.Integer
p11.setLong(100); // Parameter 1[1], will be converted to java.lang.Integer

intarray = new EXTqlist(listVlen);
intarray->addCol(fftNone,dpDefault);
intarray->addCol(fftNone,dpDefault);


intarray->insertRow();
intarray->putColVal(1, 1, p10);
intarray->putColVal(1, 2, paramType10);

intarray->insertRow();
intarray->putColVal(2, 1, p11);
intarray->putColVal(2, 2, paramType11);


// Create the Float Array.
P20.setNum(24.563); // Parameter 2[0], will be converted to java.lang.Float
P21.setNum(11.342); // Parameter 2[1], will be converted to java.lang.Float

floatarray = new EXTqlist(listVlen);
floatarray->addCol(fftNone,dpDefault);
floatarray->addCol(fftNone,dpDefault);


floatarray->insertRow();
floatarray->putColVal(1, 1, p20);
floatarray->putColVal(1, 2, paramType20);

floatarray->insertRow();
floatarray->putColVal(2, 1, p21);
floatarray->putColVal(2, 2, paramType21);

// Create the main parameter list
paramlist = new EXTqlist(listVlen);
paramlist->addCol(fftNone,dpDefault);
paramlist->addCol(fftNone,dpDefault);

// Add the Integer array to the parameter list
element1.setList(intarray,qfalse);
```

```
paramlist->insertRow();
paramlist->putColVal(1, 1, element1);
paramlist->putColVal(1, 2, paramTypeIntArr);

// Add the Float Array to the parameter list
element2.setList(floatarray,qfalse);
paramlist->insertRow();
paramlist->putColVal(2, 1, element2);
paramlist->putColVal(2, 2, paramTypeFloatArr);

//The following code uses the above list to call myfunc. For a further explanation of
//these APIs, see the section entitled “Java External Component APIs”.

jsessionid tasksession;
jobjid myobj;
str255 clsname(QTEXT(“test.store.Hello”));
str255 clspath(QTEXT(“C:\\myJava\\”));
str255 mymethod(QTEXT(“myfunc”));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
        errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,paramlist);
```

The above approach will work with all array types with the exception of character and byte arrays.

Handling Character and Byte Arrays

Character and byte arrays are converted directly from Omnis fftCharacter and fftBinary data fields. The following example demonstrates how to call the myfunc Java Method and pass an array of char.

```
Public void myfunc(char p1[])
{
…
}
```

To call the above, the following C++ is used.

```
EXTqlist *paramlist = NULL;
EXTfldval p1,paramType1;
str15 pstype(QTEXT(“[Char”));
str15 ltext(QTEXT(“Array Text”));
paramType1.setChar(pstype);


p1.setChar(ltext); // This Omnis character string will be converted directly to char[]
paramlist = new EXTqlist(listVlen);
paramlist->addCol(fftNone,dpDefault);
paramlist->addCol(fftNone,dpDefault);

paramlist->insertRow();
paramlist->putColVal(1, 1, p1);
paramlist->putColVal(1, 2, paramType1);


jsessionid tasksession;
jobjid myobj;
str255 clsname(QTEXT(“test.store.Hello”));
str255 clspath(QTEXT(“C:\\myJava\\”));
str255 mymethod(QTEXT(“myfunc”));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
        errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,paramlist);
```

The code for passing a Byte Array is similar to the above but either “[Byte” or “[cByte” must be used to specify the type and the data type of the fldval must be fftBinary.

Object and Vector Parameters

Object and Vector parameters allow arrays/lists containing a variety of different types to be passed to a java method. The following example uses an Object Array to pass a String, an Integer and a Float to a Java Method.

```
Public void myfunc(Object p1[])
{
…
}
```

To call the above, the following C++ is used.

```
EXTqlist *paramlist = NULL;
EXTqlist *objarray = NULL;
EXTfldval p1,p2,p3,fobjarray;
EXTfldval objArrType, intType, floatType, stringType;
str15 sobjArrType(QTEXT(“[cObject”));
str15 sintType(QTEXT(“Integer”));
str15 sfloatType(QTEXT(“cFloat”));
str15 sstringType(QTEXT(“cString”));
str255 ltext(QTEXT(“character string”));

objArrType.setChar(sobjArrType);
intType.setChar(sintType);
floatType.setChar(sfloatType);
stringType.setChar(sstringType);

p1.setLong(50);
p2.setNum(22.6);
p3.setChar(ltext);

objarray = new EXTqlist(listVlen);
objarray->addCol(fftNone,dpDefault);
objarray->addCol(fftNone,dpDefault);


objarray->insertRow();
objarray->putColVal(1, 1, p1);
objarray->putColVal(1, 2, intType);

objarray->insertRow();
objarray->putColVal(2, 1, p2);
objarray->putColVal(2, 2, floatType);


objarray->insertRow();
objarray->putColVal(3, 1, p3);
objarray->putColVal(3, 2, stringType);

// Create the main parameter list
paramlist = new EXTqlist(listVlen);
paramlist->addCol(fftNone,dpDefault);
paramlist->addCol(fftNone,dpDefault);

// Add the Object array to the parameter list
fobjarray.setList(objarray,qfalse);
paramlist->insertRow();
paramlist->putColVal(1, 1, fobjarray);
paramlist->putColVal(1, 2, objArrType);

//The following code uses the above list to call myfunc. For a further explanation of
//these APIs, see the section entitled “Java External Component APIs”.

jsessionid tasksession;
```

```
jobjid myobj;
str255 clsname(QTEXT(“test.store.Hello”));
str255 clspath(QTEXT(“C:\\myJava\\”));
str255 mymethod(QTEXT(“myfunc”));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
        errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,paramlist);
```

Java Objects as Parameters

In addition to the parameter types previously discussed, it is also possible to pass Java Objects as parameters. To do this, you must create a Java object using either the JPIconstructJavaObject or JPIcreateJavaObject APIs (See the section entitled “Java External Component APIs” for an explanation of these APIs). Both of these return an object ID which is used by the Omnis Java Engine (OJE) to uniquely identify an Object within its Java Session. If this ID is placed into a parameter list and the type of the list is set to “OBJECT”, the OJE will use the ID to lookup the Java Object and will pass the Object to the Java Method that you are calling.

Example:

The following example assumes that a Java Object has been previously created and that its ID is stored in a jobjid variable called myobj.

```
EXTqlist *paramlist = NULL;
EXTfldval p1,paramType1;
str15 pstype(QTEXT(“Object”));
paramType1.setChar(pstype);


p1.setLong(myobj); // Specify an existing Java Object as the parameter
paramlist = new EXTqlist(listVlen);
paramlist->addCol(fftNone,dpDefault);
paramlist->addCol(fftNone,dpDefault);

paramlist->insertRow();
paramlist->putColVal(1, 1, p1);
paramlist->putColVal(1, 2, paramType1);


jsessionid tasksession;
jobjid myobj;
str255 clsname(QTEXT(“test.store.Hello”));
str255 clspath(QTEXT(“C:\\myJava\\”));
str255 mymethod(QTEXT(“myfunc”));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
        errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,paramlist);
```

In the above example, the OJE will find the object “myobj” and will pass it as a parameter to the Java Method “myfunc”.

Return Types

The previous examples have shown how to call Java Methods in a variety of ways using different parameters. By now you should be familiar with the "Omnis Java List Structure" and how it can be used to communicate parameters to Java. Getting values back from Java works in a similar manner. The Omnis Java Engine will create an appropriate List Structure for the value that is returned. For example, if an Object Array is returned from a Java Method that contains an Integer, a Float and a String, the Omnis Java Engine will create a List Structure similar to the one exemplified in the section entitled "Object and Vector Parameters".

Example:

The following example assumes that a list called paramlist has already been constructed containing the required parameters for the "myfunc" Java Method. "myfunc" returns a Java Object Array that contains an Integer, a Float and a String.

```
qint errcode = 0;
jsessionid tasksession;
EXTfldval retval;
EXTfldval finteger;
EXTfldval ffloat
EXTfldval fstring;
jobjid myobj;
str255 clsname(QTEXT("test.store.Hello"));
str255 clspath(QTEXT("C:\\myJava\\"));
str255 mymethod(QTEXT("myfunc"));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
       errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,paramlist,&retval);
if (errcode == kOk)
{
  EXTqlist lretlist = retval.getList(qfalse);
  lretlist->getColVal(1,1,&finteger);
  lretlist->getColVal(2,1,&ffloat);
  lretlist->getColVal(3,1,&fstring);
  …
}
```

The above code pulls the Object list from the EXTfldval return value "retval" and extracts each of the array elements in turn.

It should be noted that lists that are constructed by the Omnis Java Engine for the purpose of returning values do not have a "Data Type" column. This is because the EXTfldval itself contains the Omnis type of the data being returned, so a dedicated "Data Type" column is not necessary.

Handling Java Object Return Types

It may not always be desirable to return values directly to Omnis. For example, you may have a Java Method that returns an object that cannot be translated by the Omnis Java Engine (OJE). In situations such as this, it is possible to create a Java Object to hold the contents of the return value.

Example:

```
jsessionid tasksession;
jobjid myobj;
jobjid retid;
str255 clsname(QTEXT("test.store.Hello"));
str255 clspath(QTEXT("C:\\myJava\\"));
str255 mymethod(QTEXT("myfunc"));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
        errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,paramlist,NULL,&retid);
```

In the above example, a pointer to "retid" has been passed to the callJavaMethod API. This tells callJavaMethod to create a Java Object for the value being returned. The ID of this object is then returned in "retid". This Object can then be used as a parameter to another Java Method, which can convert it to a format that is suitable for the OJE.

Java External Component Data Types

Most of the Java External Components listed in the next section make use of the following data types.

jsessionid

This ID is used to identify the Java Sessions that you create via the JPIgetCurrentJavaSession and JPIcreateJavaSession APIs. A Java Session can be thought of as an area of the Java Virtual Machine that you can use to store Java classes and objects.

Example:

```
jsessionid mysession;
mysession = JPIgetCurrentJavaSession();
```

jobjid

The jobjid type is used to identify Java Objects that are created via the JPIcreateJavaObject and JPIconstructJavaObject APIs.

Example:

```
jobjid myobj;
myobj = JPIcreateJavaObject(mysession,clsname,clspath);
```

jclassid

The jclassid type is used to identify Java classes that are created via the JPIcreateJavaClass API.

Example:

```
jclassid myclass;
myclass = JPIcreateJavaClass(mysession,clsname,clspath);
```

Java External Component APIs

The following sections provide a detailed explanation of the Omnis Java external component interface.

JPIgetCurrentJavaSession

jsessionid JPIgetCurrentJavaSession()

This API is responsible for creating Java Sessions for the current Omnis task. If a session already exists for the current Omnis task then the ID of this session is returned, otherwise a new Session is created for the current Omnis task. If this API is unable to either create or find a Java Session, the value 0 is returned.

This function will also start the Java Virtual Machine (JVM) if it has not been started already via the JPIstartJVM API.

returns - a Java Session ID or 0 if a Java Session does not exist or has not been created.

Example:

```
jsessionid tasksession;
tasksession = JPIgetCurrentJavaSession();
```

JPIcreateJavaSession

jsessionid JPIcreateJavaSession()

This API is responsible for creating Java Sessions. Sessions created via this API are not attached to any particular Omnis task and the developer is responsible for their disposal via the JPIdisposeJavaSession API. If a Java Session cannot be created, the value 0 is returned.

- **returns** – a Java Session ID or 0 if a session could not be created.

Example:

```
jsessionid mysession;
mysession = JPIcreateJavaSession();
```

JPIcreateJavaObject

jobjid JPIcreateJavaObject(jsessionid pThreadID, str255 &pClassName,
str255 &pClassPath)

The purpose of this API is to create a Java Object for a specific Java Session and a given Class Name and Class Path.

- **pThreadID** – The ID of your Java Session.
- **pClassName** – The name of the Java class that you wish to instantiate.
- **pClassPath** – The full path of your Java class, which must include a terminating path separator.
- **return** – returns the ID of the Java object that is created.

Example:

```
jsessionid tasksession;
jobjid myobj;
str255 clsname(QTEXT("test.store.Hello"));
str255 clspath(QTEXT("C:\\myJava\\"));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
```

JPIcreateJavaClass

jclassid JPIcreateJavaClass(jsessionid pThreadID, str255 &pClassName,
[str255 &pClassPath])

This API creates a Java Class object that can then be used to instantiate a Java Object with a constructor via JPIconstructJavaObject

- **pThreadID** – The ID of the Java Session that the class object is to be created for.
- **pClassName** – The name of the class that is to be created.
- **pClassPath** – The full pathname of the class. (optional).
- **return** – returns the ID of the class object that was created or 0 if the call failed.

Example:

```
jclassid myclass;
str255 clsname(QTEXT("java.lang.Integer"));
myclass = JPIcreateJavaClass(tasksession,clsname);
```

JPIconstructJavaObject

```
qlong JPIconstructJavaObject(jsessionid pThreadID, jclassid pClassID,
                             [EXTqlist *pparms], jobjid *pReturnID);
```

This API call allows Java objects to be constructed using optional parameters. This differs to JPIcreateJavaObject, which will always call the default class constructor and does not allow parameters.

- **pThreadID** – The Java Session ID.
- **pClassID** – The ID of the class that is to be instantiated as a Java object.
- **pparms** – A pointer to an EXTqlist, which contains the parameters for the Java object, that is to be constructed. (optional). For a further explanation of the format of this list, please see the section entitled "Communicating with Java".
- **pReturnID** – A pointer to a jobjid. This is used to return the ID of the object that is created if the API call is successful.
- **return** – returns a value indicating the success or failure of the operation. For further details, see the section on "Error Handling".

Example:

```
qint errcode;
jobjid myobject;
jclassid myclass;
str255 clsname(QTEXT("java.lang.Integer"));
myclass = JPIcreateJavaClass(tasksession,clsname);
errcode = JPIcreateJavaObject(tasksession,myclass,plist,&myobject);
```

where "plist" is an EXTqlist containing the parameter for the java.lang.Integer constructor. For a further explanation of the format of this list, please see the section entitled "Communicating with Java".

JPIcallJavaMethod

```
qlong JPIcallJavaMethod(jsessionid pThreadID, jobjid pObjectID,
                        str255 &pMethodName)

qlong JPIcallJavaMethod(jsessionid pThreadID, jobjid pObjectID,
                        str255 &pMethodName, EXTfldval *pRet,
                        jobjid *pRetObj = NULL)

qlong JPIcallJavaMethod(jsessionid pThreadID, jobjid pObjectID,
                        str255 &pMethodName, EXTqlist *pparms)

qlong JPIcallJavaMethod(jsessionid pThreadID, jobjid pObjectID,
                        str255 &pMethodName, EXTqlist *pparms,
                        EXTfldval *pRet, jobjid *pRetObj = NULL)
```

JPIcallJavaMethod allows you to call any method that exists in the Java object that you specify. The four variants above provide flexibility when calling Java methods as not all Java methods will have parameters or return values.

- **pThreadID** – The Java Session ID.
- **pObjectID** – The ID of the Java object that contains the method to be called.
- **pMethodName** – The Name of the method that is to be called (case sensitive).
- **pparms** – A pointer to a EXTqlist, which contains the parameters for the Java method, that is to be called. For a further explanation of the format of this list, please see the section entitled “Communicating with Java”.
- **pRet** – A pointer to an EXTfldval. After a successful call, the specified EXTfldval will contain the result of the Java method. See the section entitled “Communicating With Java” for further details.
- **pRetObj** – A pointer to a jobjid. This parameter is optional. If it is specified, then pRet must be set to NULL. Specifying pRetObj allows you to automatically create another Java object to contain the return value of your method call. After a successful call, a Java object will be created to contain your return value and the ID of the object will be returned in pRetObj.
- **return** – returns a value indicating the success or failure of the operation. For further details, see the section on “Error Handling”.

Example:

```
jsessionid tasksession;
jobjid myobj;
EXTfldval retval;
str255 clsname(QTEXT("test.store.Hello"));
str255 clspath(QTEXT("C:\\myJava\\"));
str255 mymethod(QTEXT("getCurrentDate"));

tasksesssion = JPIgetCurrentJavaSession();
myobj = JPIcreateJavaObject(tasksession,clsname,clspath );
if (tasksession && myobj)
        errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,pparms,&retval);
```

JPIgetObjectValue

qlong JPIgetObjectValue(jsessionid pThreadID, jobjid pObjectID, EXTfldval *pRet)

This API allows you to convert values from Java objects into Omnis values.

- **pThreadID** – The Java Session ID.
- **pObjectID** – The ID of the Java object whose value is to be retrieved.
- **pRet** – A pointer to an EXTfldval. After a successful call, the specified EXTfldval will contain the contents of the Java object. See the section entitled "Communicating With Java" for further details.
- **return** – returns a value indicating the success or failure of the operation. For further details, see the section on "Error Handling".

Example:

```
EXTfldval retval;
jobjid retobj;
errcode = JPIcallJavaMethod(tasksession,myobj,mymethod,pparms,NULL,&retobj);
if (errcode == kOk) errcode = JPIgetObjectValue(tasksession,retobj,&retval);
```

JPIdisposeJavaObject

void JPIdisposeJavaObject(jsessionid pThreadID, jobjid *pObjectID)

This API should be used to dispose of Java objects that were created with either JPIcreateJavaObject or JPIconstructJavaObject.

- **pThreadID** – The Java Session ID.
- **pObjectID** – The ID of the Java object that is to be disposed of.

Example:

JPIdisposeJavaObject(tasksession,&myobj);

JPIdisposeJavaClass

void JPIdisposeJavaClass(jsessionid pThreadID, jclassid *pClassID);

This API should be used to dispose of Java class objects that were created with JPIcreateJavaClass.

- **pThreadID** – The Java Session ID.
- **pClassID** – The ID of the Java class object that is to be disposed of.

Example:

JPIdisposeJavaClass(tasksession,&myclass);

JPIdisposeJavaSession

void JPIdisposeJavaSession(jsessionid *pThreadID)

This API should be used to dispose of Java Sessions that were created with JPIcreateJavaSession. It should **NOT** be used to dispose of Java Sessions that were created using JPIgetCurrentSession.

- **pThreadID** – The ID of the Java Session that is to be terminated.

Example:

JPIdisposeJavaSession(&tasksession);

JPIgetLastError

qlong JPIgetLastError(jsessionid pThreadID)

This API can be used to retrieve the error code of the most recent error reported by the Omnis Java Engine.

- **pThreadID** – The Java Session ID.

- **return** – returns an error code. For further details, see the section on "Error Handling".

Example:

```
qint errcode = 0;
errcode = JPIgetLastError(tasksession);
```

JPIgetLastErrorText

void JPIgetLastErrorText(jsessionid pThreadID, EXTfldval *pRet)

This API can be used to retrieve the text error message of the last error that occurred in the Omnis Java Engine.

- **pThreadID** – The Java Session ID.

- **pRet** – After successfully calling JPIgetLastErrorText, the text is returned in this EXTfldval.

Example:

```
EXTfldval errmsg;
JPIgetLastErrorText(tasksession,&errmsg);
```

JPIgetLastJVMErrorText

void JPIgetLastJVMErrorText(jsessionid pThreadID, EXTfldval *pRet)

This API is useful for debugging Java Errors as it returns the last error reported by the Java Virtual Machine (JVM).

- **pThreadID** – The Java Session ID.
- **pRet** – After successfully calling JPIgetLastJVMErrorText, the text of the error message is returned in this EXTfldval.

Example:

```
EXTfldval errmsg;
JPIgetLastJVMErrorText(tasksession,&errmsg);
```

JPIstartJVM

qbool JPIstartJVM()

This API can be used to start the Java Virtual Machine.

- **return** – returns true if the JVM was started successfully.

JPIisJVMrunning

qbool JPIisJVMrunning();

This API can be used to determine if the Java Virtual Machine (JVM) is already running.

- **return** – returns true if the JVM has already started.

JPIisJCoreLoaded

qbool JPIisJCoreLoaded();

This API can be used to determine if the Omnis Java Engine has been loaded.

- **return** – returns true if the JavaCore external component has been loaded

Error Handling

When programming in Java, it is not uncommon for errors to be incorrectly reported. For example, even though an exception has occurred in your Java program, it may continue to execute resulting in other misleading error message being reported later on. As debugging Java programs can be difficult when they are embedded, it is recommended that you develop and test your Java programs separately before calling them from Omnis.

If during the development process, you encounter an Omnis Java API error that does not seem to make sense, try calling the JPIgetLastJVMError API to get the last error that was reported by the Java Virtual Machine (JVM). This will tell you if the error message that you are receiving now was originally caused by a Java exception earlier on.

Error Codes

The following is a list of all of the errors that may be returned from the Omnis Java Engine (OJE).

Omnis Java Engine API Errors

The following errors may be returned from Omnis Java APIs.

```
const qint kOk =  1;
const qint kBadIntegerConstructor = -1;
const qint kFindBaseClassErr  = -2;
const qint kNoTypesSpecified  = -3;
const qint kInvalidParms = -4; // returned if parm list is invalid.
const qint kCouldNotGetTYPEfield  = -5; // Internal Error: Failed to get
java.lang.Integer.TYPE
const qint kCouldNotGetTYPEclass  = -6; // Internal Error: Failed to get
java.lang.Integer.TYPE class.
const qint kBadDoubleConstructor  = -7;
const qint kDoubleConversionFailure = -8; // Could not convert double to jfloat.
const qint kConversionFailure = -8;
const qint kBadFloatConstructor = -9;
const qint kObjCreateFail = -10;
const qint kBadCharConstructor = -11;
const qint kBadBooleanConstructor = -12;
const qint kCouldNotCreateJString = -13;
const qint kInvalidObjectID = -14;
const qint kObjectNotFound = -15;
const qint kJavaCoreNotLoaded = -16;
const qint kJNIerr = -17;
const qint kNoObjectID  = -18;
const qint kNoClassID  = -19;
const qint kLongConversionFailure = -20;
const qint kUnsupportedType = -21;
const qint kUnknownJavaType = -22;
const qint kJVMnotStarted = -23;
```

Omnis Java Engine Errors

The following errors are returned from the Java portion of the OJE and can be retrieved via the JPIgetLastError and JPIgetLastErrorText APIs.

```
const qint kNoError = 1;
const qint kNoJavaThreadID = -101;
const qint kINoState = -102;
const qint kNoThreadStart = -103;
const qint kINoSleep = -104;
const qint kINoOpCode = -105;
const qint kINoInvokeOpCode = -106;
const qint kNoClassLoader = -107;
const qint kNoClassFound = -108;
const qint kNoObjInst  = -109;
const qint kNoConstruct = -110;
const qint kNoInvokeConstruct = -111;
const qint kNoObject = -112;
const qint kNoMethod = -113;
const qint kNoInvokeMethod = -114;
```

Current Limitations

At present, the Omnis Java Engine is only capable of converting to and from the following types. Other data types may be introduced in the future.

Supported OJE Data Type conversions

Omnis Data Type	Java Data Type
fftInteger	int long short java.lang.Integer java.lang.Long java.lang.Short
fftNumber	float double java.lang.Float java.lang.Double
fftCharacter	char java.lang.Character char[] java.lang.Character[] java.lang.String
fftBinary	byte java.lang.Byte byte[] java.lang.Byte[]
fftDate	java.util.Date